
Universidade de Aveiro Departamento de Electrónica e Telecomunicações
2005

Fernando Manuel Rosmaninho Morgado Ferrão Dias

Técnicas de controlo não-linear baseadas em Redes Neuronais: do algoritmo à implementação

Universidade de Aveiro Departamento de Electrónica e Telecomunicações
2005

Fernando Manuel Rosmaninho Morgado Ferrão Dias

Técnicas de controlo não-linear baseadas em Redes Neuronais: do algoritmo à implementação

dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Engenharia Electrotécnica, realizada sob a orientação científica do Dr. Alexandre Manuel Mota, Professor Associado do Departamento de Engenharia Electrónica e Telecomunicações da Universidade de Aveiro

o júri

presidente

Prof. Doutor Carlos Alberto Diogo Soares Borrego
professor catedrático da Universidade de Aveiro por delegação da Reitora da Universidade de Aveiro

Prof. Doutor João Manuel Lage de Miranda Lemos
professor catedrático do Instituto Superior Técnico da Universidade Técnica de Lisboa

Prof. Doutor António Eduardo de Barros Ruano
professor associado com agregação da Faculdade de Ciências e Tecnologia da Universidade do Algarve

Prof. Doutor José Alberto Gouveia Fonseca
professor associado da Universidade de Aveiro

Prof. Doutor Alexandre Manuel Moutela Nunes da Mota
professor associado da Universidade de Aveiro (Orientador)

Prof. Doutor Tomás António Mendes Oliveira e Silva
professor associado da Universidade de Aveiro

agradecimentos

Em primeiro lugar, gostaria de agradecer ao meu orientador, Professor Doutor Alexandre Manuel Mota, pela sua capacidade de trabalho, sentido crítico e grande disponibilidade. Ele é não só o mentor deste trabalho, como também o grande responsável pelo bom termo a que o mesmo chegou.
À Ana, companheira de vida, colega de trabalho, revisora e crítica de toda esta tese.
À minha Mãe, Nininha, e à minha irmã Guida pelas revisões da tese e dos artigos, respectivamente.
Ao José António Vieira, pelo que me obrigou a repensar e a discutir e pelo que produzimos em conjunto. A sua contribuição foi excelente.
Aos meus colegas e amigos pela ajudas e apoio que me deram ao longo destes anos. Contei com eles para amenizar as dificuldades e muitas vezes também para encontrar soluções.
Em Aveiro: Victor e Susana, Teixeira, Mário, Salvador, Ricardo e Maria João, Paulo Neves, Luís Almeida, Valter Silva, Paulo Bartolomeu e Pedro Duarte.
Em Setúbal: Armando Pires, Miguel Moreira, José Carvalho, Isabel, Fernando e Gisela e Eduardo e Susana.
Aos meus alunos Pedro Ferreira e Pedro Ribeiro, pelo trabalho que desenvolvemos.
A todos eles cabe uma parte deste trabalho e todos contribuíram para que fosse um pouco mais fácil chegar aqui.

resumo

O presente trabalho analisa soluções de controlo não-linear baseadas em Redes Neuronais e apresenta a sua aplicação a um caso prático, desde o algoritmo de treino até à implementação física em *hardware*.

O estudo inicial do estado da arte da utilização das Redes Neuronais para o controlo leva à proposta de soluções iterativas para a definição da arquitectura das mesmas e para o estudo das técnicas de Regularização e Paragem de Treino Antecipada, através dos Algoritmos Genéticos e à proposta de uma forma de validação dos modelos obtidos.

Ao longo da tese são utilizadas quatro malhas para o controlo baseado em modelos, uma das quais uma contribuição original, e é implementado um processo de identificação *on-line*, tendo por base o algoritmo de treino Levenberg-Marquardt e a técnica de Paragem de Treino Antecipada que permite o controlo de um sistema, sem necessidade de recorrer ao conhecimento prévio das suas características.

O trabalho é finalizado com um estudo do *hardware* comercial disponível para a implementação de Redes Neuronais e com o desenvolvimento de uma solução de *hardware* utilizando uma FPGA.

De referir que o trabalho prático de teste das soluções apresentadas é realizado com dados reais provenientes de um forno eléctrico de escala reduzida.

abstract

The present work analyses non-linear control solutions based on Neural Networks and presents its application to a case study, from the training algorithm to the hardware implementation.
The initial study of the state of the art of Neural Networks use in control led to a proposal of iterative solutions for architecture definition and establishment of the Regularization and Early Stopping parameters, through the use of Genetic Algorithms and to the proposal of a new validation technique for the models.
Throughout this thesis, four different loops for model based control are used, one of which is an original contribution, and an on-line identification procedure, based on the Levenberg-Marquardt algorithm with Early Stopping that allows system identification without previous knowledge of its characteristics.
The work is finalized with a commercial hardware study for Neural Networks and with the development of a hardware solution based on a FPGA.
It is worth mentioning that proposed solutions are tested with real data provided by a reduced scale electric kiln.

apoios

Este trabalho foi apoiado pelas seguintes instituições:

Escola Superior de Tecnologia de Setúbal e Instituto Politécnico de Setúbal: apoio financeiro concedido com vista à participação em conferências para a apresentação de resultados parciais obtidos no âmbito deste trabalho e dispensa de serviço docente concedida para a conclusão desta tese.

Fundação da Ciência e Tecnologia, programa Fundo de Apoio à Comunidade Científica: apoio financeiro concedido com vista à participação em algumas conferências para a apresentação de resultados parciais obtidos no âmbito desta tese.

Universidade de Aveiro: pelas condições para o desenvolvimento do trabalho realizado no âmbito desta tese.

TESE DE DOUTORAMENTO

Fernando Morgado Dias

1 de Março de 2005

Conteúdo

1 Introdução **1**
- 1.1 Organização da tese . . . 2
- 1.2 Contribuição desta tese . . . 4

2 Redes Neuronais **5**
- 2.1 Introdução . . . 5
- 2.2 Inspiração biológica . . . 6
- 2.3 Perspectiva histórica . . . 8
- 2.4 Redes Neuronais sem realimentação . . . 9
- 2.5 Redes Neuronais com realimentação . . . 12
- 2.6 Outros tipos de Redes Neuronais . . . 12
- 2.7 Algoritmos de treino . . . 14
  - 2.7.1 Optimização baseada em derivadas . . . 15
  - 2.7.2 Optimização sem derivadas . . . 23
  - 2.7.3 Tipos de implementação dos algoritmos . . . 23
- 2.8 Estado da arte . . . 25
  - 2.8.1 Novos tipos de Redes Neuronais . . . 25
  - 2.8.2 Algoritmos de treino . . . 26
  - 2.8.3 Outros tópicos . . . 27
- 2.9 Conclusão . . . 27

3 Identificação de sistemas com Redes Neuronais **29**
- 3.1 Introdução . . . 29
- 3.2 Aquisição de dados . . . 30
  - 3.2.1 Condições experimentais . . . 31
  - 3.2.2 Escolha do período de amostragem e estudo preliminar do sistema 31
  - 3.2.3 Escolha do tipo de dados a utilizar . . . 32
- 3.3 Preparação dos dados . . . 32
  - 3.3.1 Escalar os dados . . . 32
  - 3.3.2 Pré-processamento dos dados . . . 34
  - 3.3.3 Filtragem . . . 34
  - 3.3.4 Divisão dos dados em diversos conjuntos . . . 35
- 3.4 Classes de modelos . . . 35
  - 3.4.1 Classes de modelos lineares . . . 35

3.4.2 Classes de modelos não-lineares baseados em Redes Neuronais . 39
3.5 Ordem do sistema 40
3.6 Estruturas de treino 42
3.6.1 Modelo directo 42
3.6.2 Modelo inverso 43
3.7 Capacidade de generalizar 51
3.7.1 Qualidade do modelo 51
3.7.2 Regularização 53
3.7.3 Paragem de Treino Antecipada 54
3.7.4 Outras técnicas 55
3.8 Estado da arte 56
3.9 Conclusão 57

4 Redes Neuronais e Algoritmos Genéticos 59
4.1 Introdução 59
4.2 Algoritmos Genéticos 61
4.2.1 Cruzamento 62
4.2.2 Mutação 62
4.2.3 Elitismo 63
4.2.4 Função de aptidão 63
4.2.5 Algoritmo 64
4.2.6 Outros operadores 65
4.3 Redes Neuronais e Algoritmos Genéticos 66
4.3.1 Estratégias de codificação 67
4.4 Conclusão 68

5 Estruturas de controlo 69
5.1 Introdução 69
5.2 Controlador com Modelo Inverso 69
5.3 *Controller Output Error Method e Indirect Inverse Adaptation* 72
5.4 Controlador Aditivo 72
5.4.1 Controlador Aditivo puro 74
5.4.2 Controlador Aditivo misto 75
5.5 Controlador Baseado em Modelo Interno 75
5.5.1 Controlador Baseado em Modelo Interno com modelos neuronais 76
5.6 Controlador Aditivo Baseado em Modelo Interno 78
5.7 Conclusão 81

6 Um caso prático 83
6.1 Introdução 83
6.2 O sistema 83
6.2.1 Módulo de potência 85
6.3 Escolha do ambiente MATLAB 87
6.4 Norma SCPI 88

6.4.1 Função SCPI 88
6.5 Aquisição de dados 96
6.5.1 Escolha do período de amostragem 96
6.5.2 Resposta em malha aberta 96
6.5.3 Escolha do tipo de dados a utilizar 98
6.6 Preparação dos dados 98
6.7 Classes de modelos 99
6.8 "Ordem" do sistema 99
6.9 Estruturas de treino 100
6.10 Capacidade de generalizar 100
6.11 Modelos lineares 100
6.11.1 Modelos baseados em Redes Neuronais 102
6.12 Automatização do processo de optimização de modelos 103
6.12.1 Introdução 104
6.12.2 Detalhes de implementação 104
6.12.3 Optimização com AGs 105
6.13 Conclusões 109

7 Resultados de controlo 113
7.1 Introdução 113
7.2 Modelos optimizados pelo operador humano 113
7.3 Modelos optimizados com Paragem de Treino Antecipada 118
7.4 Modelos optimizados com Regularização 119
7.5 Controlo com identificação *on-line* 121
7.5.1 Introdução 121
7.5.2 Outras implementações *on-line* 122
7.6 Outros trabalhos 134
7.7 Conclusões 134

8 Implementação de Redes Neuronais em *hardware* 137
8.1 Introdução 137
8.2 *Hardware* Comercial 138
8.2.1 Sumário 138
8.2.2 Introdução 139
8.2.3 Especificação do *hardware* 140
8.2.4 Tipos de implementação 141
8.2.5 *Hardware* comercial 141
8.2.6 Dificuldades encontradas neste trabalho 148
8.2.7 Conclusões 149
8.3 Implementação usando uma FPGA 150
8.3.1 Sumário 150
8.3.2 Introdução 150
8.3.3 Implementação de *hardware* 152
8.3.4 Sistema de teste 156

8.3.5 Resultados . . . 157
8.3.6 Conclusões e trabalho futuro . . . 158
8.3.7 Agradecimento . . . 160
8.4 Conclusão . . . 160

9 Conclusões e trabalho futuro 163
9.1 Conclusões . . . 163
9.1.1 Técnica híbrida genérica/especializada . . . 164
9.1.2 Controlo Aditivo Baseado em Modelo Interno . . . 164
9.1.3 Implementação do algoritmo Levenberg-Marquardt em janela deslizante com Paragem de Treino Antecipada . . . 164
9.1.4 Implementação em *hardware* de uma RN com uma FPGA. . . . 164
9.1.5 Qualidade de controlo . . . 164
9.2 Trabalho futuro . . . 165
9.2.1 Sistemas de tempo real . . . 165
9.2.2 *Hardware* . . . 166
9.2.3 Influência de *jitter* . . . 167

Anexo A - Glossário de termos e abreviaturas 183

Anexo B - Exemplo de código de implementação de uma Rede Neuronal 185

Anexo C - Lista de publicações 189

Lista de Figuras

2.1 Ilustração de um neurónio natural simplificado. 7
2.2 Exemplo de RN sem realimentação. Da esquerda para a direita: camada de entrada, camada escondida e camada de saída. 9
2.3 Representação de uma RN sob a forma matricial. 10
2.4 Exemplo de um neurónio. 10
2.5 Exemplo de RN com realimentação. 13
2.6 Exemplo de uma má escolha de α_k. 17
2.7 RN de uma camada escondida em representação de forma a facilitar a análise matricial. 22
2.8 Ilustração dos processos de treino em grupo e do treino recursivo. . . . 24

3.1 Procedimento de identificação de sistemas. 30
3.2 Aplicação do factor de escala aos modelos. 33
3.3 Exemplo de um modelo da classe NNARX. 39
3.4 Exemplo de um modelo da classe NNARMAX. 39
3.5 Exemplo de um modelo da classe NNFIR. 40
3.6 Exemplo de um modelo da classe NNOE. 40
3.7 Exemplo da evolução da função de custo em função do número de regressores do modelo. 41
3.8 Exemplo da evolução da função de custo em função do número de regressores do modelo para um sistema afectado por ruído. 42
3.9 Diagrama de blocos da estrutura de treino do modelo directo. 43
3.10 Diagrama de blocos da estrutura de treino usada para criar modelos inversos genéricos. 44
3.11 Diagrama de blocos da estrutura de treino de um modelo inverso especializado. 45
3.12 Representação do processo de treino de um modelo inverso especializado com inclusão de um modelo de referência. 47
3.13 Outra representação possível para o treino de um modelo inverso especializado. 47
3.14 Diagrama de blocos do *Indirect Inverse Adaptation.* 48
3.15 Diagrama de blocos do *Controller Output Error Method.* 49
3.16 Ilustração da situação que origina o dilema deslocamento-variância. . . 53

4.1 Exemplo da codificação de um indivíduo numa optimização com AG e a respectiva interpretação da informação. 61
4.2 Exemplo do operador de cruzamento. Na metade superior num só ponto, na parte inferior em dois pontos. 62
4.3 Exemplo da operação de mutação. 63
4.4 Funcionamento do 3ºpasso da proposta de implementação de AG. . . . 64
4.5 Exemplo do operador de conjugação. 65

5.1 Controlador com Modelo Inverso com detalhe sobre as entradas do modelo inverso. $r(k)$ é a referência, ou seja o comportamento que se pretende para o sistema em malha fechada, $u(k)$ o sinal de controlo e $y(k)$ a saída do sistema a controlar. 70
5.2 Diagrama de blocos do modelo de um sistema de 2ª Ordem. 70
5.3 Diagrama de blocos do modelo inverso de um sistema de 2ª Ordem. . . 71
5.4 Diagrama de blocos do modelo inverso de um sistema de 2ª Ordem, com substituição das amostras de y não disponíveis por amostras da referência. 71
5.5 Exemplo de um Controlador com modelo inverso, usando um modelo de 2ª ordem. 71
5.6 Diagrama de blocos do *Indirect Inverse Adaptation.* 72
5.7 Diagrama de blocos representativo do *Controller Output Error Method.* 73
5.8 Estrutura usada para o Controlo Aditivo. Os sinais $r(k)$, $u(k)$ e $y(k)$ têm o mesmo significado que anteriormente. 73
5.9 Diagrama de blocos de um Controlador Aditivo Puro. 74
5.10 Diagrama de blocos de um Controlador Aditivo Misto. 75
5.11 Estrutura clássica de um Controlador Baseado em Modelo Interno. O sinal $\hat{y}$(k) é a estimativa do sinal de saída gerada pelo modelo directo do sistema e e(k) o erro entre a saída do sistema e a estimativa $\hat{y}$. 76
5.12 Estrutura do IMC com detalhe de implementação dos modelos directo e inverso. 77
5.13 Estrutura do Controlador Aditivo Baseado em Modelo Interno. 78
5.14 AIMC representado de forma genérica. 80

6.1 Vista esquemática do forno. 84
6.2 Vista exterior do forno. 84
6.3 Diagrama de blocos da malha de identificação e controlo. 85
6.4 Vista do *Data Logger.* 86
6.5 Diagrama de blocos do módulo de potência. 86
6.6 Implementação física do módulo de potência. 87
6.7 Resposta do sistema em malha aberta. 97
6.8 Característica estática do sistema sem coincidência total entre o aquecimento e o arrefecimento. 98
6.9 Sinal utilizado na preparação dos modelos neuronais, 75% dos pontos (os iniciais) foram utilizados como sequência de treino e os restantes como sequência de teste. 99

6.10 Sinal escalado utilizado na preparação dos modelos neuronais. 100
6.11 Resultados de simulação do modelo linear de 1ª ordem. 101
6.12 Resultados de simulação do modelo linear de 2ª ordem. 102
6.13 Resultados de simulação do modelo neuronal optimizado com recurso aos conhecimentos do operador humano. 103
6.14 Diagrama de blocos do sistema de optimização com AGs. 106
6.15 Resultados de simulação do modelo neuronal optimizado com Algoritmos Genéticos utilizando a técnica de Paragem de Treino Antecipada (par_2). 107
6.16 Resultados de simulação do modelo neuronal optimizado com Algoritmos Genéticos utilizando a técnica de Regularização (par_3). 108
6.17 Representação gráfica da evolução da população para o modelo directo treinado com *Early Stopping*. 109
6.18 Representação gráfica da evolução da população para o modelo inverso treinado com *Early Stopping*. 110
6.19 Representação gráfica da evolução da população para o modelo directo treinado com Regularização. 110
6.20 Representação gráfica da evolução da população para o modelo inverso treinado com Regularização. 111

7.1 Resultado de Controlo com Modelo Inverso (DIC) utilizando modelos optimizados pelo operador humano. 114
7.2 Resultado de Controlo Baseado em Modelo Interno (IMC) utilizando modelos optimizados pelo operador humano. 115
7.3 Resultado de Controlo Aditivo (AFC). 116
7.4 Resultado de Controlo Aditivo Baseado em Modelo Interno utilizando modelos optimizados pelo operador humano (AIMC). 117
7.5 Controlo do sistema de teste por intermédio do controlador PI. 118
7.6 Resultado de Controlo com Modelo Inverso (DIC) utilizando modelos optimizados com Paragem de Treino Antecipada. 119
7.7 Resultado de Controlo Baseado em Modelo Interno (IMC) utilizando modelos optimizados com Paragem de Treino Antecipada. 120
7.8 Resultado de Controlo com Modelo Inverso (DIC) utilizando modelos optimizados com Regularização. 121
7.9 Resultado de Controlo Baseado em Modelo Interno (IMC) utilizando modelos optimizados com Regularização. 122
7.10 Conteúdo da janela deslizante em função da evolução das épocas. . . . 123
7.11 Diagrama de blocos representativo do algoritmo de identificação on-line. 125
7.12 Disposição das janelas de treino e de teste, com indicação do sentido de deslocamento das amostras. 126
7.13 Resultado de identificação *on-line* e Controlo com Modelo Inverso (DIC). A fase inicial de controlo é assegurada por um PI. 127
7.14 Resultado de identificação *on-line* e Controlo Baseado em Modelo Interno (IMC). A fase inicial de controlo é assegurada por um PI. 128

7.15 Resultado de identificação *on-line* e Controlo Aditivo Baseado em Modelo Interno (AIMC). A fase inicial de controlo é assegurada por um PI. 129
7.16 Resultado de identificação *on-line* e Controlo com Modelo Inverso (DIC), utilizando janela de teste deslizante. A fase inicial de controlo é assegurada por um PI. 131
7.17 Resultado de identificação *on-line* e Controlo Baseado em Modelo Interno (IMC), utilizando janela de teste deslizante. A fase inicial de controlo é assegurada por um PI. 132
7.18 Resultado de identificação *on-line* e Controlo Aditivo Baseado em Modelo Interno (AIMC), utilizando janela de teste deslizante. A fase inicial de controlo é assegurada por um PI. 133

8.1 Classificação do *hardware* para RNs em categorias. 142
8.2 Imagem do *kit* do cyclone SmartPack. 154
8.3 Estrutura lógica do bloco de processamento básico no ANNP 155
8.4 Janela principal da aplicação de teste e correcção de erros. 156
8.5 Método de teste da FPGA. 157
8.6 Diagrama de blocos para o controlo directo inverso. 157
8.7 Resultados do controlo directo inverso utilizando a FPGA com o sinal da rampa. 158
8.8 Resultados do controlo directo inverso utilizando a FPGA com o sinal de rampa e de onda quadrada. 159
8.9 Resultados do controlo directo inverso utilizando a FPGA com o sinal pseudo-aleatório. 160

Lista de Tabelas

2.1 Funções de activação frequentemente usadas na implementação de Redes Neuronais. 11
2.2 Tipos de Redes Neuronais 13
2.3 Tipos de Redes Neuronais 14

3.1 Comparação das estruturas de treino existentes. 50

5.1 Modos de funcionamento da malha genérica AIMC 80

6.1 Operações com SCPI - Tempos Envolvidos - 9600*bps* 93
6.2 Operações com SCPI - Tempos Envolvidos - 19200*bps* 93
6.3 Visualização Gráfica dos Resultados - Tempos Envolvidos 95
6.4 Temperaturas finais obtidas nos testes em malha aberta. 96
6.5 Subidas de temperatura resultantes dos diversos degraus de tensão na entrada. 97
6.6 Número de bits e gamas de variação permitidas para os parâmetros. . . 105
6.7 Número de bits e gamas de variação permitidas para os parâmetros. . . 108

7.1 Sumário dos valores de EQM para modelos optimizados pelo operador humano. 117
7.2 Sumário dos valores de EQM para modelos optimizados com Paragem de Treino Antecipada. 119
7.3 Sumário dos valores de EQM para modelos optimizados com Regularização. 120
7.4 Sumário dos valores de EQM para os testes de identificação e controlo on-line, com janela de teste deslizante. 131

8.1 Implementações analógicas de RNs. 143
8.2 Implementações digitais com arquitectura *slice*. 145
8.3 Implementações digitais com arquitectura SIMD. 146
8.4 Implementações digitais com arquitectura *systolic array*. 146
8.5 Implementações digitais com arquitectura RBF. 147
8.6 Implementações digitais com outras arquitecturas. 148
8.7 Implementações analógicas de RNs. 149
8.8 Resumo das características dos modelos inversos testados na FPGA. . . 157

8.9 Comparação entre os resultados obtidos com a FPGA e o MATLAB em termos de erro quadrático médio. 159

Capítulo 1

Introdução

"An expert is a person who has made all the mistakes that can be made in a very narrow field." - Niels Bohr, cientista. (1885 - 1962)

O presente trabalho surge no âmbito de um projecto de investigação com o objectivo de desenvolver um forno com elevado grau de controlo dos perfis de temperatura para a indústria cerâmica e do vidro. Atendendo ao elevado grau de precisão pretendido e à possível degradação dos elementos que compõem o forno é necessário que a instrumentação associada permita o controlo em tempo-real e a identificação *on-line* pelo que há que desenvolver adequadamente os sensores, a electrónica de tratamento dos sinais e os algoritmos de controlo.

Devido ao comportamento não-linear do sistema entendeu-se que as Redes Neuronais seriam uma potencial solução a explorar. Pretende-se portanto fazer um estudo das soluções existentes no campo das Redes Neuronais para a identificação e controlo de sistemas de forma a obter tanto controladores baseados em redes neuronais, dentro das estruturas convencionais de controlo, como obter processos que permitam a identificação de sistemas *on-line* e o respectivo controlo. O objectivo do trabalho passou ainda por desenvolver uma implementação completa que fosse do *hardware* do sistema ao *hardware* do controlador, ou seja, utilizando um sistema real, construir a malha completa.

O estudo inicial do estado da arte da utilização das Redes Neuronais para o controlo levou à proposta de soluções iterativas para a definição da arquitectura das mesmas e para o estudo das técnicas de Regularização e Paragem de Treino Antecipada, através dos Algoritmos Genéticos, e à proposta de uma forma de validação dos modelos obtidos.

Ao longo da tese são utilizadas quatro malhas para o controlo baseado em modelos, uma das quais uma contribuição original, e é implementado um processo de identificação *on-line*, tendo por base o algoritmo de treino Levenberg-Marquardt e a técnica de Paragem de Treino Antecipada que permite o controlo de um sistema, sem necessidade de recorrer ao conhecimento prévio das suas características.

O trabalho é finalizado com um estudo do *hardware* comercial disponível para a implementação de Redes Neuronais e com o desenvolvimento de uma solução de *hardware* utilizando uma FPGA. De referir que o trabalho prático de teste das soluções

apresentadas é realizado com dados reais provenientes de um forno eléctrico de escala reduzida.

1.1 Organização da tese

A tese está organizada nos seguintes capítulos: Introdução, Redes Neuronais, Identificação de sistemas com Redes Neuronais, Redes Neuronais e Algoritmos Genéticos, Estruturas de controlo, Um caso prático, Resultados de Controlo, *Hardware* para Redes Neuronais e Conclusões.

No primeiro capítulo faz-se uma curta introdução e explica-se a organização da tese.

No segundo capítulo pretende-se fazer um resumo da história, inspiração biológica, tipos de RNs e algoritmos de treino que são utilizados com as RNs para identificação e controlo de sistemas.

O terceiro capítulo está dedicado às questões relativas ao processo que leva à identificação de um sistema. Do tema deste capítulo foi publicado o seguinte artigo:

- "A new hybrid direct/specialized approach for generating inverse neural models", Fernando Morgado Dias, Ana Antunes, Alexandre Mota, WSEAS Transactions on Systems, Issue 4, Vol 3, pp 1521-1529, Junho de 2004.

O quarto capítulo faz uma introdução à teoria associada aos algoritmos genéticos e à sua aplicação no âmbito das RNs. As publicações relacionadas com este capítulo são:

- "Regularization versus early stopping: a case study with a real system", Fernando Morgado Dias, Ana Antunes, Alexandre Manuel Mota, 2^{nd} IFAC Conference Control Systems Design (CSD'03), Bratislava, República Eslovaca, 2003.
- "Automating the Construction of Neural Models for Control Purposes using Genetic Algorithms", Fernando Morgado Dias, Ana Antunes, Alexandre Manuel Mota, 28^{th} Annual Conference of the IEEE Industrial Electronics Society, Sevilha, Espanha, 2002.

No quinto capítulo são analisadas algumas estruturas que são habitualmente utilizadas para fazer o controlo a partir de modelos feitos com RNs e introduzida uma nova estrutura criada pelo autor. As publicações relacionadas com este capítulo são:

- "Additive Internal Model Control: an Application with Neural Models in a Kiln", Fernando Morgado Dias, Ana Antunes, Alexandre Manuel Mota, 28^{th} Annual Conference of the IEEE Industrial Electronics Society, Sevilha, Espanha, 2002.
- "A Comparison between a PID and Internal Model Control using Neural Networks", Fernando Morgado Dias e Alexandre Manuel Mota, 5^{th} World Multi-Conference on Systemics, Cybernetics and Informatics, volume V, pp.268-273, Orlando, EUA, 2001.

- "Comparison between different Control Strategies using Neural Networks", Fernando Morgado Dias e Alexandre Manuel Mota, 9^{th} Mediterranean Conference on Control and Automation, Dubrovnik, Croácia, 2001.

- "Additive Feedforward Control of a Kiln Using Neural Networks", Fernando Morgado Dias e Alexandre Manuel Mota, IASTED International Conference on Modelling, Identification, and Control, Innsbruck Austria, 2001.

- "Direct Inverse Control of a Kiln", Fernando Morgado Dias e Alexandre Manuel Mota, 4^{th} Portuguese Conference on Automatic Control, Guimarães, 2000.

No sexto capítulo é apresentado um caso prático, um sistema real que será utilizado como teste para as soluções desenvolvidas.

O sétimo capítulo apresenta os resultados obtidos com o sistema abordado no capítulo anterior. Neste capítulo é abordado a identificação *on-line* e desta parte do capítulo foi publicado o seguinte artigo:

- "Implementing the Levenberg-Marquardt Algorithm on-line: a Sliding Window Approach with Early Stopping", Fernando Morgado Dias, Ana Antunes, José Vieira, Alexandre Manuel Mota, 2^{nd} IFAC Workshop on Advanced Fuzzy/Neural Control, Setembro de 2004.

O oitavo capítulo é dedicado ao *hardware* para RNs. Neste capítulo é feita uma análise de mercado relativa às soluções comercialmente existentes e é apresentada uma implementação de controlo em *hardware*. As publicações relacionadas com este capítulo são:

- "Artificial Neural Networks: a Review of Commercial Hardware", Fernando Morgado Dias, Ana Antunes, Alexandre Mota, aceite para publicação na revista Engineering Applications of Artificial Intelligence.

- "Artificial Neural Networks Processor - a Hardware Implementation using a FPGA", Pedro Ferreira, Pedro Ribeiro, Ana Antunes, Fernando Morgado Dias, Field-Programmable Logic and its Applications, Setembro de 2004.

- "Commercial Hardware for Artificial Neural Networks: a Survey", Fernando Morgado Dias, Ana Antunes, Alexandre Manuel Mota, SICICA - 5^{th} IFAC International Symposium on Inteligent Components and Instruments for Control Applications, Aveiro, 2003.

No nono capítulo são retiradas as conclusões e delineado o trabalho futuro.

1.2 Contribuição desta tese

A presente tese contém diversas contribuições inovadoras que estão dispersas ao longo de vários capítulos. A primeira dessas contribuições surge no capítulo três, sobre Redes Neuronais e diz respeito a uma técnica híbrida de treino de modelos inversos, designada por Técnica Híbrida genérica/especializada para modelos inversos, que utiliza uma simulação de controlo como forma de avaliar a qualidade dos modelos inversos e permite ultrapassar as limitações existentes nas outras soluções.

A segunda contribuição surge no capítulo cinco, dedicado às estruturas de controlo e é relativo à malha de Controlo Aditivo Baseado em Modelo Interno. Esta malha que conjuga o funcionamento do Controlador Aditivo e do Controlador Baseado em Modelo Interno é uma contribuição original do autor desta tese.

A terceira contribuição diz respeito à implementação do algoritmo de Levenberg-Marquardt em janela deslizante com Paragem de Treino Antecipada e está no capítulo sete.

No capítulo oito é possível encontrar um estudo do *hardware* disponível comercialmente para a implementação de Redes Neuronais e a última contribuição original: a implementação de uma Rede Neuronal em FPGA. Desta parte do trabalho deve salientar-se a elevada resolução e a qualidade da solução utilizada na aproximação usada na implementação da tangente hiperbólica.

Capítulo 2

Redes Neuronais

"É com cepticismo que encaro a presunção da ciência relativamente à sua objectividade e ao seu carácter definitivo. Tenho dificuldade em aceitar que os resultados científicos, principalmente em neurobiologia, sejam algo mais do que aproximações provisórias para serem saboreadas por uns tempos e abandonadas logo que surjam melhores explicações."
- António R. Damásio, em O Erro de Descartes, cientista. (1948 -)

2.1 Introdução

O mundo actual é caracterizado pela existência de um cada vez maior número de processos de controlo automático, quer seja resultado da simples procura de maior conforto, quer seja pela necessidade imperiosa dos ambientes industriais. No primeiro caso pode pensar-se, por exemplo, nos edifícios inteligentes onde se optimizam os consumos energéticos e memorizam as preferências do utilizador e no segundo caso na substituição do operador humano pela máquina em ambientes hostis à vida humana.

O controlo automático tem assumido diversas formas em função da sua própria evolução, sendo possível distinguir a sua existência desde as primeiras obras do génio humano. Um simples autoclismo é, na sua essência, um sistema de controlo automático, uma vez que permite o enchimento de água de um recipiente até um determinado nível pré-definido sem a intervenção humana.

Para sistemas um pouco mais complexos é normalmente necessário estudar o seu comportamento de forma a desenvolver um modelo que os represente e produzir um controlador para que, sem a intervenção de um operador humano, o conjunto controlador e sistema desempenhe a sua função da forma pretendida.

Existem habitualmente duas formas de estabelecer o modelo que representa um sistema: a descrição matemática desse sistema através das equações que correspondem aos princípios físicos do seu funcionamento ou a utilização de dados sobre o sistema, recolhidos através de experiências práticas, que permitam inferir esse modelo. A primeira opção requer, muitas vezes, demasiado tempo e no caso de sistemas mais complexos pode tornar-se demasiadamente difícil. O segundo caso, habitualmente designado por identificação de sistemas, pode evitar estas dificuldades, necessitando no entanto de

uma recolha de dados sobre o sistema em questão por forma a evidenciar as suas características nas diversas gamas de funcionamento [1].

Quando o controlo não é feito de forma automática é utilizado um operador humano que além de ter a capacidade de memorizar o funcionamento do sistema, estabelece mentalmente um modelo do seu comportamento. Este tipo de informação é guardada no cérebro cujas unidades base de processamento são os neurónios. Se o operador humano estabelece com facilidade modelos, memoriza pontos de funcionamento dos sistemas e fá-lo através do seu cérebro, utilizando os neurónios, então faz sentido tentar perceber de que forma os neurónios guardam essa informação e se é possível imitá-la. Este deverá ter sido o pensamento dos primeiros investigadores que se dedicaram à área das Redes Neuronais (RNs), ainda que este tipo de tentativa dê origem, com alguma frequência, a erros. Um bom exemplo desta situação são as tentativas de construir máquinas voadoras com base na imitação do voo das aves.

Naturalmente este tipo de conhecimento serve não só para o estabelecimento de modelos como para a criação de controladores. Actualmente, já com algumas décadas de investigação, a área de Redes Neuronais, que se dedica à construção de modelos artificiais simplificados dos neurónios e à sua utilização em conjuntos por forma a emular tarefas que o cérebro humano desempenha, constitui um universo tão vasto que é difícil ter uma visão completa, estando a sua aplicação espalhada por um vasto leque de valências científicas.

As RNs Artificiais, referidas ao longo deste documento apenas como RNs, têm actualmente uma inspiração biológica remota nos neurónios que constituem o cérebro. As RNs utilizadas com objectivos de modelização são o resultado da pesquisa científica de várias décadas que sofreu avanços e recuos como será documentado, de forma breve, na secção dedicada à perspectiva histórica.

Do ponto de vista da topologia, existem apenas dois tipos de RN: com realimentação e sem realimentação [2]. Ambos são abordados neste capítulo, onde é feita uma introdução às RNs, da inspiração biológica até à forma como acontece a aprendizagem efectuada nas RNs.

2.2 Inspiração biológica

O cérebro é constituído por um elevado número de neurónios com elevado número de ligações. Do ponto de vista de interesse para este trabalho os neurónios têm três componentes: as dendrites, o corpo do neurónio e o axónio ou cilindro-eixo [3].

As dendrites formam redes ramificadas de fibras nervosas com capacidade de transportar sinais eléctricos até ao corpo do neurónio.

O corpo do neurónio desempenha as funções de soma das entradas afectadas pelos respectivos pesos e de aplicação de uma função não linear de limiar sobre os sinais que recebe.

O axónio é uma fibra nervosa única que transporta o sinal de saída do neurónio até outros neurónios. O ponto de contacto entre o axónio de um neurónio e a dendrite de outro neurónio é designado por sinapse.

Os elementos do neurónio simplificado que foram referidos estão ilustrados na figura 2.1.

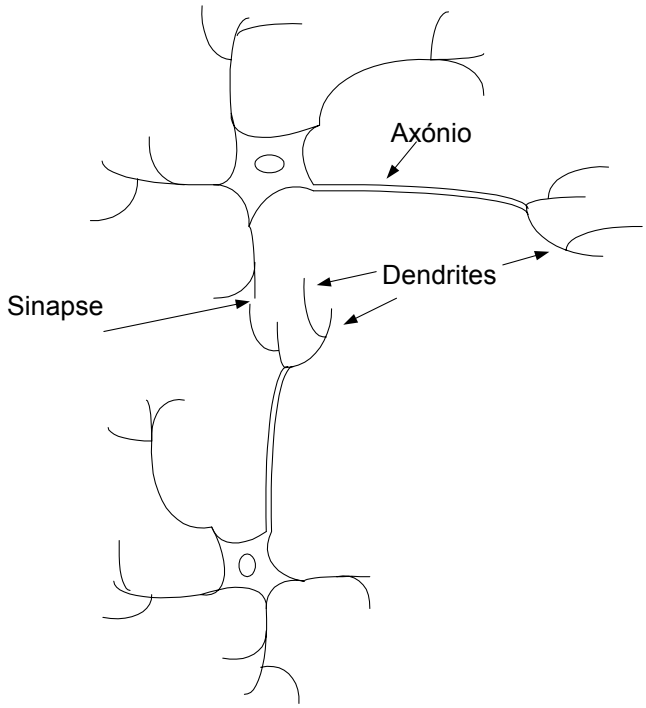


Figura 2.1: Ilustração de um neurónio natural simplificado.

A função desempenhada por cada neurónio é determinada pelas ligações aos restantes neurónios, pelo peso associado às sinapses e por um processo químico complexo [4].

O cérebro humano é constituído por cerca de 10 biliões de neurónios com cerca de 10 triliões de sinapses. Cada neurónio possui, em média, cerca de 1000 sinapses, embora alguns possam ter 5000 ou 6000 [5].

Algumas funções dentro da estrutura neurológica estão já definidas à nascença sendo que outras vão sendo definidas com as experiências ocorridas na vida. Um exemplo interessante citado em [4] relata que um gato jovem cuja utilização de um olho lhe tenha sido impedida durante um período crítico do seu desenvolvimento, nunca chegará a desenvolver visão normal nesse olho.

Os neurónios utilizados nas RNs artificiais não são mais que um modelo simplificado dos neurónios existentes na natureza [6] e o próprio processo de aprendizagem é feito de forma semelhante ao natural: aprendizagem através de exemplos.

Apesar da elevada velocidade dos actuais computadores comparada com a dos neurónios naturais (10^{-9}s comparada com 10^{-3}s [4]) o cérebro é capaz de efectuar tarefas muito mais rapidamente do que um computador e existem muitas tarefas que mesmo utilizando RNs artificiais ainda não conseguem ser desempenhadas pelos computadores. Esta situação é devida essencialmente à estrutura paralela dos neurónios

no cérebro enquanto um computador desempenha tarefas de forma sequencial.

2.3 Perspectiva histórica

O início da visão moderna das RNs é habitualmente [4], [7] e [8] atribuído ao trabalho de McCulloch e Pitts em 1943, que estudou o potencial de redes de neurónios simplificados, mostrando que este tipo de redes tinha capacidade de desempenhar qualquer função lógica [8].

Hebb, em 1949, fez um importante trabalho ao estudar os fenómenos de adaptação das RNs biológicas, acabando por propor a regra de aprendizagem que ficou conhecida como Regra de Hebb [4].

Minsky (1951) criou um computador neuronal designado por Snarl, que constituiu um dos primeiros modelos reais de redes neuronais [8].

Rosenblatt (1958) criou o Perceptrão (do inglês *Perceptron*), semelhante ao neurónio utilizado neste trabalho mas utilizando como função de activação apenas a função de Heaviside que origina uma saída do tipo binário e uma regra de aprendizagem [4].

Quase ao mesmo tempo Widrow e Hoff introduziram um novo algoritmo de aprendizagem e utilizaram-no para treinar RNs com função de activação linear. A regra de aprendizagem de Widrow-Hoff ainda hoje é utilizada [4].

Em 1969 foi publicado o livro *Perceptrons* de Minsky e Papert que acabou por constituir um sério revés para o desenvolvimento das RNs. Minsky e Papert introduziram uma análise rigorosa do perceptrão, provando várias propriedades e identificando limitações [7]. Entre as limitações ficou famosa a incapacidade de um perceptrão isolado desempenhar a função ou exclusivo.

Na verdade, ainda hoje, a capacidade individual dos neurónios utilizados é bastante elementar, o que não é impeditivo que uma RN desempenhe globalmente funções importantes mas estes resultados levaram a que parte da comunidade científica suspendesse a investigação na área das RNs [4].

A dificuldade em estabelecer algoritmos capazes de fazer a aprendizagem de RNs mais complexas e a inexistência de capacidade computacional disponível levou então ao adiamento da investigação na área das RNs.

No entanto, alguns trabalhos continuaram a ser desenvolvidos durante a década de 70: Kohonen (1972) desenvolveu um novo tipo de RNs que servem de memórias e ficaram conhecidas como Redes de Kohonen e Grossberg (1976) desenvolveu o que seriam chamadas redes auto-organizadas (do inglês *self-organizing networks*) [4].

Na década de 80, com o aparecimento dos computadores pessoais e o avanço das estações de trabalho foram criadas condições para o retomar da investigação nesta área.

Dois trabalhos seriam, no entanto, determinantes para esta retoma [4]: o desenvolvimento das memórias associativas de Hopfield (1982) e o desenvolvimento do algoritmo *Backpropagation*, atribuído ao trabalho independente de vários pesquisadores, sendo destacado o trabalho de Rumelhart e McClelland (1986).

A partir desta fase muitos resultados e aplicações têm sido desenvolvidos nas mais diversas áreas, do controlo à medicina, passando pela economia, agricultura e meteoro-

logia. Particularmente importante para o presente trabalho são os artigos de Cybenko [9], Hornik [10] e [11] entre outros, por provarem que as RNs são aproximadores universais. Nestes trabalhos pode distinguir-se, no entanto, dois tipos de resultados [12]:

- dado o número de neurónios necessários, ou seja a estrutura da RN, usando funções de activação contínuas e diferenciáveis e com apenas uma camada escondida, as RNs são capazes de aproximar qualquer função contínua [9] [10].

- dado o número de neurónios necessários, ou seja a estrutura da RN, usando funções de activação contínuas e diferenciáveis e com duas camadas escondidas, as RNs são capazes de aproximar qualquer função [13] [14].

Apesar do volume de trabalho que actualmente continua a ser desenvolvido, não é evidente até onde pode ir esta área até porque a capacidade potencial estará ainda longe de ser atingida. Para ser possível avaliar o que está feito é interessante a citação do relatório de pesquisa americano DARPA feita em [8] que estima (em 1989) que o computador de maior capacidade existente terá no máximo capacidade para simular o cérebro de uma mosca. É evidente que hoje esta capacidade será superior mas ainda há muito por fazer.

2.4 Redes Neuronais sem realimentação

As Redes Neuronais sem realimentação (do inglês *Feedforward Neural Networks* - FNN) são redes constituídas habitualmente por elementos base iguais, os neurónios, que estão dispostos em camadas e ligados de forma a que o sinal flua da entrada para a saída sem realimentação e sem ligações laterais. Um exemplo deste tipo de RN pode ser visto na figura 2.2.

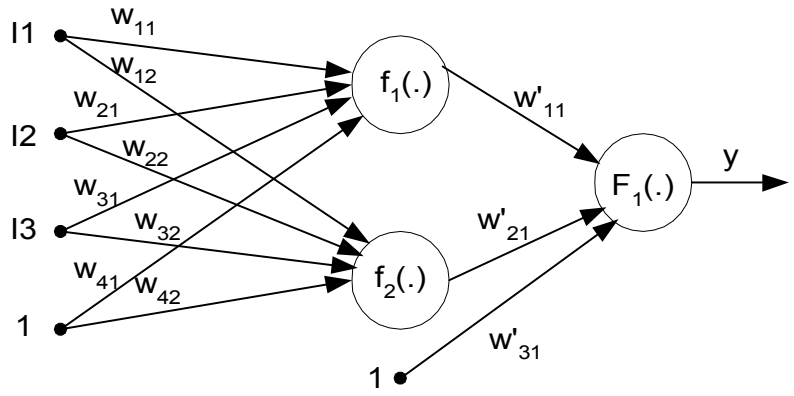


Figura 2.2: Exemplo de RN sem realimentação. Da esquerda para a direita: camada de entrada, camada escondida e camada de saída.

Esta forma de representar a RN, onde são postas mais em evidência as ligações e a associação dos pesos, é apenas uma das várias possibilidades. Uma forma alternativa está ilustrada na figura 2.3. Nesta figura, a RN foi desenhada por forma a facilitar a compreensão da representação da RN sob a forma matricial [15].

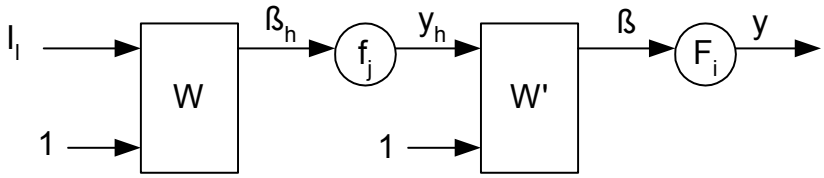


Figura 2.3: Representação de uma RN sob a forma matricial.

Numa rede neuronal é habitual agruparem-se os elementos em camadas. Na notação que será utilizada ao longo deste texto será designada por camada de entrada o conjunto das entradas sem neurónios, por camada escondida a camada cujas saídas não estão acessíveis directamente (onde estão as funções f_1 e f_2) e por camada de saída a camada em que os neurónios estão directamente ligados às saídas da rede neuronal (onde está F_1). A entrada unitária é habitualmente designada por entrada de deslocamento (do inglês *bias*) das funções de activação e não é usada nas designações da arquitectura da rede. Em função desta notação uma RN como o exemplo da figura 2.2 pode ser designada por 3-2-1 (3 entradas, 2 neurónios na camada escondida e 1 neurónio na saída). Embora estes exemplos contenham apenas uma camada escondida, não existe limite para o número destas camadas.

O detalhe da função implementada pelo neurónio pode ser visto na figura 2.4. Esta representação está feita do ponto de vista funcional, estando a entrada de deslocamento ligada à função de activação e não ao somador. Isto serve para mostrar que, apesar de efectivamente a entrada de deslocamento ser processada pelo somador, o seu efeito pode ser analisado como um deslocamento da função de activação do neurónio.

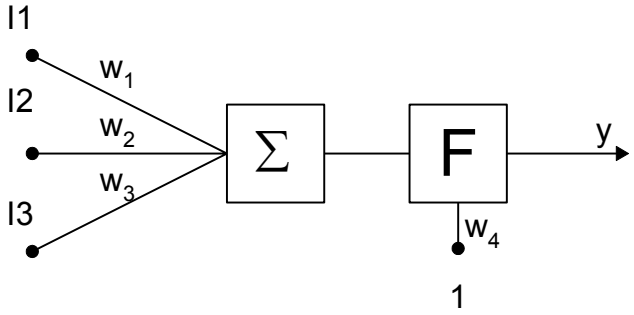


Figura 2.4: Exemplo de um neurónio.

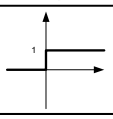
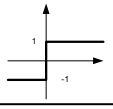
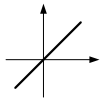
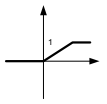
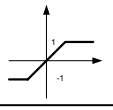
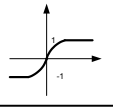
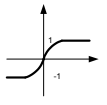
Nome	**Função**	**Função**
Heaviside	x<0 → y=0 x≥0 → y=1	
Heaviside simétrico	x<0 → y=-1 x≥0 → y=1	
Linear	y=x	
Linear com saturação	x<0 → y=0 0<x<1 → y=x x>1 → y=1	
Linear simétrico com saturação	x<-1 → y=0 -1<x<1 → y=x x>1 → y=1	
Log-sigmoidal	$y = \frac{1}{1+e^{-x}}$	
Tangente hiperbólica	$y = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	

Tabela 2.1: Funções de activação frequentemente usadas na implementação de Redes Neuronais.

Um neurónio genérico implementa a função:

$$y = F(\sum_{i=1}^{n} I_i w_i) \tag{2.1}$$

onde I_i representa a i-ésima entrada e w_i o peso correspondente e F representa a função de activação que deve ser diferenciável ou não diferenciável apenas num conjunto finito de pontos. Na equação 2.1 a entrada de deslocamento encontra-se englobada em I_i. Alguns exemplos de funções frequentemente utilizadas em RNs são apresentadas na tabela 2.1 [4]. Numa grande parte das aplicações a função de activação dos neurónios da camada de saída é uma função linear (ver tabela 2.1) por permitir que sejam obtidos valores finais numa gama não circunscrita.

Todas as funções apresentadas, com excepção da linear, incluem algum tipo de não linearidade.

Este tipo de RN, no seu todo, implementa a função:

$$y = F(\sum_{j=1}^{nn} w'_{jl} f_j(\sum_{l=1}^{ne} w_{lj} I_l)) \tag{2.2}$$

onde *nn* representa o número de neurónios da camada escondida e *ne* representa o número de entradas.

Trata-se de uma função composta de produtos e somas, onde a cada ligação está associado um peso, cujo principal mérito provém da capacidade de aproximar funções. Como foi demonstrado em [9], [10] e [11], as RN podem aproximar qualquer função com a precisão pretendida, desde que disponham do número de neurónios necessários. O facto de as funções utilizadas não serem lineares facilita o seu uso para a aproximação de funções não lineares.

Na grande maioria das implementações utilizam-se RNs com apenas uma camada escondida. Não deve, no entanto, ser ignorado o resultado apresentado em [16] onde é demonstrado que existem funções, especialmente quando se pretende criar modelos inversos, que apenas podem ser correctamente aproximadas por RNs com duas camadas escondidas. Este resultado não impede, no entanto, que em muitos casos concretos seja possível utilizar RNs com apenas uma camada escondida.

No caso concreto das RN sem realimentação o seu comportamento pode ser descrito como realizando um mapeamento estático entre a entrada e a saída. No entanto, caso nas entradas existam valores anteriores de entradas e saídas, é introduzido na RN o efeito de memória dinâmica e a RN pode ser utilizada para o mapeamento de sistemas dinâmicos.

2.5 Redes Neuronais com realimentação

As RNs com realimentação são RNs que incluem, pelo menos, uma ligação de uma camada mais próxima da saída para uma camada menos próxima da saída ou uma ligação entre neurónios da mesma camada. A figura 2.5 contém um exemplo de uma RN de uma camada escondida com realimentação.

O atraso representado na figura 2.5 significa que a saída da RN é utilizada como entrada da mesma RN após o tempo definido por esse atraso. Desta forma a RN com realimentação guarda informação do seu próprio estado interno o que facilita a representação de sistemas dinâmicos.

Este tipo de RNs necessita no entanto de métodos de aprendizagem mais complexos do que os que são utilizados para as RNs sem realimentação [17].

2.6 Outros tipos de Redes Neuronais

Apesar de não fazerem parte do objecto de estudo deste trabalho vale a pena referir a existência de diversos outros tipos de RNs, sendo que alguns são tão antigos como as FNNs e que outros apareceram mais recentemente. As tabelas 2.2 e 2.3 apresentam, de forma resumida e necessariamente não exaustiva, alguns tipos de RNs comparados com as FNNs relativamente a algumas das suas características [18].

Nestas tabelas RBF significa *Radial Basis Function* e ART representa *Adaptive Resonance Theory.* As redes de Hopfield e Boltzmann devem os seus nomes aos investigadores que iniciaram o seu estudo.

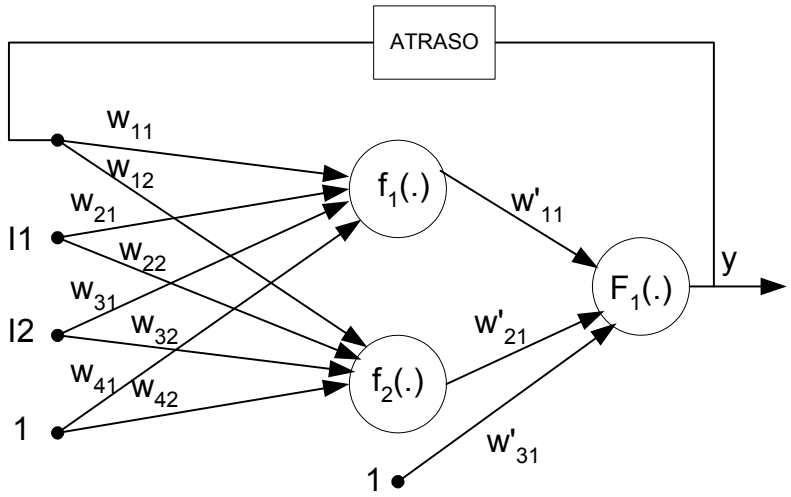


Figura 2.5: Exemplo de RN com realimentação.

Tipo de Rede	Função de Activação	Ligações
FNNs	ver tabela 2.1	multi-camada sem realimentação
RBF	gaussiana e linear	multi-camada sem realimentação
Hopfield	sigmoide e Heaviside	realimentada
Boltzmann	probabilística	realimentada
ART	sinal máximo	realimentada

Tabela 2.2: Tipos de Redes Neuronais

A tabela 2.2 apresenta informação relativa às funções de activação e ao tipo de ligações existentes na estrutura destas RNs.

A tabela 2.3 contém informação sobre o tipo de mecanismo de aprendizagem, também designado por algoritmo de treino e sobre as áreas mais comuns de aplicação. Os mecanismos de aprendizagem são designados por supervisionados quando existem exemplos da saída pretendida para a RN que podem ser usados para verificar se ela está a desempenhar a função pretendida. São designados por não supervisionados quando se verifica a ausência desses exemplos. Neste último caso a aprendizagem é feita apenas com base nas entradas. Este processo pode parecer difícil de realizar uma vez que não se tem conhecimento do que é pretendido que a RN faça, mas é possível fazer com que as RNs aprendam a separar as entradas em classes [4]. Alguns documentos referem o mecanismo de aprendizagem *Reinforcement Learning* que é semelhante ao método supervisionado, sendo que neste caso em vez de exemplos existem notas para classificar a saída produzida pela RN [4].

As áreas de aplicação são apenas introduzidas de forma indicativa, uma vez que quase todos os tipos de RNs já foram utilizados para todas as áreas de aplicações [18].

No presente trabalho pretendeu-se utilizar as RNs para efectuar controlo, sendo as redes mais indicadas as RBFs e as FNNs. Estes dois tipos de RNs têm capacidades

Tipo de Rede	Mecanismo de Aprendizagem	Aplicação
FNNs	Supervisionado	Controlo , classificação, optimização, predição e reconhecimento de padrões
RBF	Supervisionado ou híbrido	Controlo , previsão
Hopfield	Supervisionado ou não supervisionado	Memória associativa e optimização
Boltzmann	Estocástico	Optimização
ART	Competitivo	Reconhecimento de padrões

Tabela 2.3: Tipos de Redes Neuronais

semelhantes ao nível dos algoritmos de treino desenvolvidos e das potencialidades que apresentam.

Estes dois tipos de RNs têm muito mais em comum do que a maioria da literatura parece sugerir. A única diferença fundamental está na forma como os neurónios das camadas escondidas combinam os valores provenientes das camadas anteriores da RN [2]. A opção de utilizar as FNNs resulta da sua maior divulgação ao nível das ferramentas desenvolvidas de livre utilização, que permitiram implementar modelos de forma rápida.

2.7 Algoritmos de treino

Se a RN é um aproximador universal, como é feito o ajuste dos parâmetros que permitem essa aproximação?

Os parâmetros em causa são os pesos que estão associados a cada ligação e ao processo de ajuste dos mesmos é vulgar designar-se por treino. O treino é um processo iterativo em que após cada alteração dos pesos se avalia o desempenho do novo conjunto e se procura uma nova solução caso a actual não seja ainda satisfatória. Alguns destes processos ou algoritmos foram desenvolvidos especificamente para as RNs (*Backpropagation*) e outros foram adaptados (Mínimos quadrados e Levenberg-Marquardt). A existência de parâmetros ajustáveis leva a que as RNs sejam por vezes designadas por Redes Adaptativas (do inglês *Adaptive Networks*) [17].

Antes de iniciar o processo de treino é necessário dispor de informação sobre o sistema que se pretende modelizar. Essa informação pode existir de uma forma qualitativa ou quantitativa. No caso das aplicações incluídas neste trabalho a informação é exclusivamente quantitativa e existe na forma de pares de valores do tipo (entradas, saídas). Estes pares no seu conjunto servem a dupla função de exemplos para o treino e para o teste de avaliação do desempenho da RN. Este teste torna-se necessário porque, frequentemente, o comportamento da RN perante a sequência de treino é diferente do comportamento perante uma nova sequência.

Detalhes sobre a fase de recolha de informação e sobre a separação dos dados em conjuntos de treino e de teste encontram-se no capítulo 3.

Associado aos processos de treino existe a necessidade de avaliar a qualidade da solução encontrada, muitas vezes mesmo durante o treino. Nos casos que serão descritos a avaliação será feita com base no Erro Quadrático (EQ) ou no Erro Quadrático Médio (EQM)[1]. Considerando um sistema de uma única saída e várias entradas (vulgarmente designados por MISO - *Multi Input Single Output*), por uma questão de simplicidade, embora sem perda de generalidade, pode escrever-se:

$$EQ = \sum_{i=1}^{N} (o - y)^2 \tag{2.3}$$

$$EQM = \frac{1}{N}. \sum_{i=1}^{N} (o - y)^2 \tag{2.4}$$

onde o é a saída pretendida, y é a saída obtida com a RN e N o número de pontos utilizados.

Os algoritmos de treino utilizados em RNs podem-se dividir consoante o tipo de procura da melhor solução, ou seja, se utilizam ou não derivadas no seu processo de optimização.

2.7.1 Optimização baseada em derivadas

Neste grupo encontram-se a maioria dos métodos utilizados para o treino das RNs: *steepest descent*, Newton, Quasi-Newton, Gauss-Newton e Levenberg-Marquardt. Esta secção não pretende ser exaustiva pelo que serão descritos apenas os métodos que foram usados no presente trabalho e alguns outros necessários para a melhor compreensão dos mesmos.

Partindo de uma função qualquer $F(x)$ (de que são exemplos as funções 2.3 e 2.4) que é usada como índice de performance (IP), pretende-se optimizar um comportamento, o que normalmente se traduz por minimizar o IP [4]. As técnicas de optimização descritas aqui pressupõem que não existe uma forma analítica de determinar o(s) mínimo(s) do IP pelo que se torna necessário recorrer a uma forma iterativa de procura ou simplesmente tentar obter uma aproximação para esse mínimo.

De forma genérica, partindo de uma estimativa inicial x_0 e actualizando-a em cada iteração de acordo com uma equação da forma:

$$x_{k+1} = x_k + \alpha_k.p_k \tag{2.5}$$

ou

$$\triangle x_k = (x_{k+1} - x_k) = \alpha_k.p_k \tag{2.6}$$

[1]Em rigor, o EQM é $E[(o-y)^2]$. Para N constante não há diferença entre os mínimos das equações 2.3 e 2.4.

onde k representa o índice da iteração, p_k representa uma direcção de procura e α_k a taxa de aprendizagem que determina o comprimento do passo dado em cada iteração.

A escolha de p_k é o factor que distingue os algoritmos que serão apresentados (com excepção do *Backpropagation*).

Steepest descent

O algoritmo de *steepest descent*, que em português poderia ser designado por direcção do gradiente negativo mais íngreme ou máxima inclinação descendente, é frequentemente designado na literatura como *Backpropagation*. No entanto, o *Backpropagation* é, de facto, a forma de relacionar o erro na saída de uma RN com a correcção que deve ser efectuada em cada peso, sendo independente da direcção de procura escolhida, ou seja o *Backpropagation* é essencialmente a regra de derivação em cadeia que relaciona o erro na saída da RN com a mudança que deve ser efectuada no peso.

Em cada iteração quando se actualiza o valor de x_{k+1}, o objectivo é, para o caso das equações 2.3 e 2.4, obter um valor do IP mais baixo do que o da iteração precedente:

$$F(x_{k+1}) < F(x_k) \tag{2.7}$$

De que forma é possível encontrar p_k que, para α_k suficientemente pequeno, faça diminuir o IP?

A expansão em série de Taylor de primeira ordem de $F(x)$ em torno da iteração x_k é:

$$F(x_{k+1}) = F(x_k + \triangle x_k) \simeq F(x_k) + G(x,k).\triangle x_k \tag{2.8}$$

onde $G(x,k) = \nabla F(x)|_{x=x_k}$ é o gradiente de $F(x)|_{x=x_k}$.

Para que $F(x_{k+1})$ seja menor do que $F(x_k)$, o produto $G(x,k).\triangle x_k$, tem que ser menor do que zero. Como $G(x,k).\triangle x_k = G(x,k).\alpha_k.p_k$ e α_k é sempre positivo, isto implica que $G(x,k).p_k < 0$. Esta condição porém não é, só por si, suficiente para garantir que $F(x_{k+1})$ seja menor do que $F(x_k)$, como está ilustrado no exemplo da secção 2.7.1.

Qual é a direcção que leva à maior descida do valor de $F(x)$ (*steepest descent*)?

Sendo p_k uma direcção podemos pressupor que o seu módulo não varia, então o produto interno do gradiente $G(x,k)$ por p_k terá o seu valor máximo negativo quando $p_k = -G(x,k)$.

A equação de actualização da estimativa do mínimo da função $F(x)$ será então:

$$x_{k+1} = x_k - \alpha_k.G(x,k) \tag{2.9}$$

Esta equação é extremamente intuitiva uma vez que corresponde a procurar em cada iteração a direcção que fornece o valor mais negativo da derivada e usar essa direcção como indicador para a minimização da função.

No entanto a existência do factor α_k obriga a uma escolha criteriosa do seu valor porque se for demasiado elevado pode originar uma subida do valor de F(x) e se for demasiado baixo leva ao não aproveitamento de todo o potencial de descida.

Escolha da taxa de aprendizagem α_k A influência da escolha de α_k pode ser ilustrada com o exemplo muito simples a duas dimensões que está representado na figura 2.6.

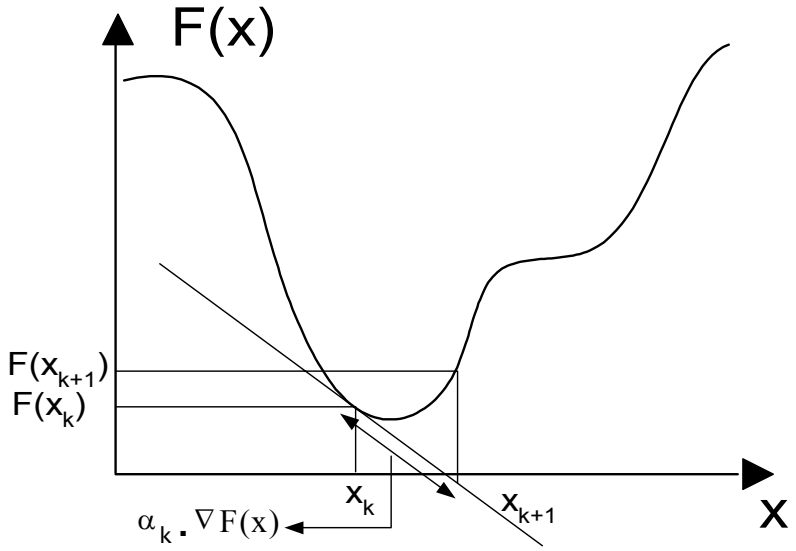


Figura 2.6: Exemplo de uma má escolha de α_k.

No exemplo da figura 2.6 podemos ver a recta tangente à curva que representa a função F(x), no ponto x_k.

Esta recta dá a direcção da derivada nesse ponto e a nova iteração x_{k+1} depende do valor de α_k que for escolhido.

Neste exemplo o valor é demasiado elevado dando origem a que $F(x_{k+1})$ seja maior do que $F(x_k)$. De forma inversa a escolha de um valor demasiado baixo dá origem a uma convergência demasiado lenta.

Exemplificada que está a importância de α_k importa saber como escolher o valor a usar. Uma possibilidade consiste em escolher um valor fixo bastante baixo, por exemplo $0.001 < \alpha_k < 0.1$ [15], ainda que este valor seja arbitrário e dependente da escala da função.

Como a convergência perto de um mínimo pode ser mais difícil e é suposto com as iterações do *steepest descent* F(x) aproximar-se de um mínimo, é possível escolher α_k variável, por exemplo $\alpha_k = 1/k$ [4].

Existem também possibilidades de uma escolha adaptativa. Como exemplo é apresentado um procedimento de minimização da função unidimensional definida segundo a direcção de p_k, que permite a actualização do valor de α em cada iteração em função do resultado obtido nessa iteração:

```
α = valor_inicial     // valores habituais 0.001 < α < 0.1
α_incremento = 1.05
α_decremento = 0.7
```

```
for i=1:  numero_de_iterações
  if F(xk+1) > F(xk)
     ignorar iteração
     α = α * α_decremento
    else
       aceitar a iteração
       α = α * α_incremento
  end
end
```

Neste procedimento é definido um valor inicial para o parâmetro α, um valor para o incremento e para o decremento. Em cada iteração se não foi possível minimizar o valor de $F(x)$, em relação à iteração anterior, rejeita-se a iteração e aplica-se um decremento ao valor de α, caso contrário aceita-se a iteração e aplica-se o factor de incremento ao valor de α.

Método de Newton

O método do *steepest descent* parte da expansão em série de Taylor truncada para primeira ordem, enquanto o método de Newton utiliza a série de Taylor truncada para segunda ordem:

$$F(x_{k+1}) = F(x_k + \triangle x_k) \simeq F(x_k) + G(x,k).\triangle x_k + \frac{1}{2}.\triangle x_k.H(x,k).\triangle x_k \tag{2.10}$$

onde $G(x,k)$ é o gradiente de $F(x)$ e $H(x,k) = \nabla^2 F(x)|_{x=x_k}$é a matriz Hessiana de $F(x)|_{x=x_k}$.

A equação 2.10 define a função $F(x_{k+1})$. Como qualquer outra função pode ser derivada (neste caso em relação a $\triangle x_k$) e igualada a zero para localizar um extremo (se a matriz Hessiana for definida positiva então o extremo será necessariamente um mínimo):

$$G(x,k) + H(x,k).\triangle x_k = 0 \tag{2.11}$$

então pode obter-se $\triangle x_k$:

$$\triangle x_k = -H(x,k)^{-1}.G(x,k) \tag{2.12}$$

A regra de iteração do método de Newton será então:

$$x_{k+1} = x_k - H(x,k)^{-1}.G(x,k) \tag{2.13}$$

Este método permite sempre encontrar um mínimo para funções quadráticas numa só iteração. Se a função não for quadrática a convergência será mais lenta, não existindo garantia de convergência [4].

Gauss-Newton e Levenberg-Marquardt

Os algoritmos de Gauss-Newton e Levenberg-Marquardt são variações do método de Newton, que foi criado para minimizar funções que são somas de quadrados de outras funções não lineares [4]. O algoritmo denominado Levenberg-Marquardt é devido aos trabalhos realizados, de forma independente, pelos autores em [19] e [20], respectivamente, e foi posteriormente adaptado para a aplicação no âmbito das RNs, sendo [21] uma das primeiras aplicações deste algoritmo às RNs. A aplicação às RNs é explicada em detalhe por muitos autores, como por exemplo [22], [4] e [1].

Considerando novamente uma função $F(x)$ como o IP que se pretende minimizar, mas partindo do pressuposto que se trate de um somatório de funções quadráticas [4] como será o caso das equações 2.3 e 2.4 quando a função a minimizar corresponder à saída de um sistema do tipo *Multi Input Single Output* (MISO), por uma questão de simplicidade, embora sem perda de generalidade, a saída será expressa em forma vectorial e $F(x)$ será (simplificando a notação por supressão do índice relativo à iteração):

$$F(x) = \sum_{i=1}^{N} v_i^2(x) \tag{2.14}$$

onde $v_i(x) = o_i - y_i$.

O j-ésimo elemento do gradiente será:

$$G(x)_j = 2.\sum_{i=1}^{N} v_i(x).\frac{\partial v_i(x)}{\partial x_j} \tag{2.15}$$

portanto o gradiente pode ser escrito matricialmente da seguinte forma:

$$G(x) = 2.J^T(x).v(x) \tag{2.16}$$

onde $J(x)$ representa a matriz Jacobiana.

Para o método de Newton é usada, para além do gradiente, a matriz Hessiana. O seu k,j-ésimo elemento (correspondente à linha k, coluna j) para funções quadráticas pode ser expresso do seguinte modo:

$$H(x)_{k,j} = 2.\sum_{i=1}^{N}\left[\frac{\partial v_i(x)}{\partial x_k}.\frac{\partial v_i(x)}{\partial x_j} + v_i(x).\frac{\partial^2 v_i(x)}{\partial x.\partial x_j}\right] \tag{2.17}$$

A matriz Hessiana expressa em forma matricial é:

$$H(x) = 2.J^T(x).J(x) + 2.S(x) \tag{2.18}$$

onde

$$S(x) = \sum_{i=1}^{N} v_i(x).\frac{\partial^2 v_i(x)}{\partial x_k.\partial x_j} \tag{2.19}$$

se for assumido que $S(x)$ é pequeno comparativamente ao primeiro termo da matriz Hessiana, é possível aproximar a matriz Hessiana pela seguinte expressão:

$$H(x) \simeq 2.J^T(x).J(x) \tag{2.20}$$

Substituindo esta expressão e a expressão 2.16 na equação 2.12 obtém-se o algoritmo de *Gauss-Newton:*

$$\triangle x_k = -\left[2.J^T(x_k).J(x_k)\right]^{-1}.2.J^T(x_k).v(x_k) \tag{2.21}$$

Uma limitação que pode surgir com este método é que a aproximação da matriz Hessiana pode não ser invertível. Uma solução possível é utilizar uma aproximação diferente:

$$Hm(x) = H(x) + \mu.I \tag{2.22}$$

onde I é a matriz identidade e μ um valor tal que torne a matriz $Hm(x)$ definida positiva e portanto invertível e $Hm(x)$ é a matriz Hessiana modificada. Esta alteração corresponde ao algoritmo de *Levenberg-Marquardt* que pode ser expresso pela equação seguinte:

$$\triangle x_k = -\left[2.J^T(x_k).J(x_k) + \mu_k I\right]^{-1}.2.J^T(x_k).v(x_k) \tag{2.23}$$

O parâmetro μ ou μ_k, para indicar que o seu valor não é necessariamente fixo, desempenha um papel extremamente importante no funcionamento do algoritmo de *Levenberg-Marquardt.* Não só garante a estabilidade do algoritmo (tornando a matriz Hm invertível) como determina a velocidade de convergência do algoritmo. Tendo em conta esta importância justifica-se a análise mais detalhada dos métodos para calcular o seu valor.

A aproximação feita com a matriz Hessiana é válida apenas numa vizinhança da corrente iteração.

Isto corresponde a procurar o valor de x que minimiza a função escolhida como IP (por exemplo as funções 2.3 ou 2.4), ou seja, $x_{k+1} = \arg\min_x (IP(x))$ mas restringido a $|x - x_k| \leqslant \delta_k$, onde δ_k representa o raio em torno do qual a aproximação é válida e aparece afectado de um índice uma vez que também este raio que representa a confiança depositada na aproximação pode ser variável consoante o algoritmo escolhido para μ_k.

Existe, obviamente, uma relação entre δ_k e μ_k, sendo que ao aumentarmos μ_k estamos a diminuir a vizinhança δ_k[23]. Como não existe uma expressão exacta que relacione os dois parâmetros, foram desenvolvidas diversas soluções [23].

A solução proposta por Fletcher [23], que é usada em [24] utiliza a expressão:

$$r_k = \frac{V_N(x_k) - V_N(x_k + p_k)}{V_N(x_k) - L_k(x_k + p_k)} \tag{2.24}$$

para obter uma medida da qualidade da aproximação. Aqui V_N é a expressão que se pretende minimizar, por exemplo 2.4, L_k é a estimativa desse valor calculada a partir da aproximação obtida pela expansão em série de Taylor de segunda ordem e p_k é a

direcção de procura, neste caso a direcção correspondente ao algoritmo de *Levenberg-Marquardt.*

Esta medida da qualidade da aproximação é depois usada na determinação de μ_k da seguinte forma:

1-Escolher os valores iniciais de x_0 e μ_0.
2-Calcular a direcção de procura p_k.
3-Se $r_k > 0.75$ então $\mu_k = \mu_k/2$.
4-Se $r_k < 0.25$ então $\mu_k = 2\mu_k$.
5-Se $V_N(x_k + p_k) < V_N(x_k)$ então a iteração é aceite.
6-Se a condição de paragem de treino ainda não foi atingida, voltar ao passo 2.

Uma importante característica do algoritmo de *Levenberg-Marquardt* é o facto de a partir do parâmetro μ_k (que pode ser variado a cada iteração) ser possível modificar o seu comportamento. Quando μ_k é aumentado e se torna dominante, o comportamento é semelhante ao *steepest descent:*

$$\triangle x_k = -\frac{2}{\mu_k}.J^T(x_k).v(x_k) \tag{2.25}$$

e para valores menores de μ_k retoma um comportamento semelhante ao método de Newton.

Em geral, o *steepest descent* converge, mesmo para estimativas iniciais más, mas requer um longo tempo de convergência. No caso do algoritmo Gauss-Newton, as soluções convergem rapidamente, mas exigem boas estimativas iniciais. Usando um μ_k adaptativo, como foi proposto para α_k no método do *steepest descent,* é possível obter uma solução de compromisso entre a velocidade do método de Newton e a garantia de convergência do *steepest descent* [4].

Backpropagation

Embora sendo referido muitas vezes na literatura como um algoritmo de optimização o *Backpropagation* é na realidade mais do que isso: é o processo de, a partir do erro na saída, calcular o erro referido a cada neurónio da rede neuronal. Ou seja, é o processo de propagar o erro em sentido inverso: da saída para a entrada. Este processo é utilizado por todos os algoritmos quando aplicados a RNs uma vez que é necessário conhecer o valor do erro à saída de cada neurónio para poder alterar os pesos relativos a esse neurónio.

O algoritmo habitualmente designado por *Backpropagation* é na realidade uma aplicação de *steepest descent* e da regra de derivação em cadeia.

Retomando a equação 2.9 e usando como IP a função da equação 2.3, a aplicação do *steepest descent* ao caso das redes neuronais resulta na seguinte equação genérica para os pesos:

$$w(k+1) = \ w(k) - \alpha.\frac{\partial EQ}{\partial w} \tag{2.26}$$

Ao contrário do que acontecia com o algoritmo do *steepest descent* o cálculo da derivada $\frac{\partial EQ}{\partial w}$ não é directo porque os diversos pesos que é necessário actualizar encontram-se no interior das diferentes camadas que compõem a RN.

No caso concreto das RNs com uma camada escondida os pesos desta camada dependem da camada de saída e a aplicação das equações anteriores não é idêntica em cada uma das camadas.

Como calcular a derivada $\frac{\partial EQ}{\partial w}$?

Na figura 2.7 podemos ver uma RN com uma camada escondida representada de forma a facilitar a análise em formato matricial.

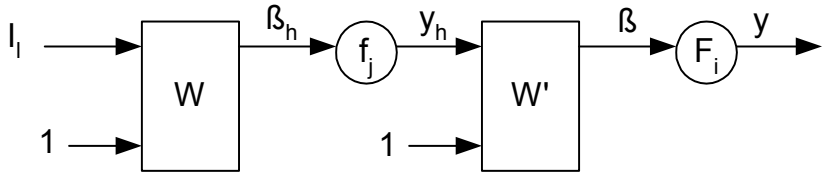


Figura 2.7: RN de uma camada escondida em representação de forma a facilitar a análise matricial.

Retomando a equação 2.2:

$$y = F(\sum_{j=1}^{nn} w'_{jl}.f_j(\sum_{l=1}^{ne} w_{lj}.I_l))$$

que descreve a função implementada pela RN, designando por y_h a saída da camada escondida (a saída global é y) e usando a regra da derivação em cadeia, pode escrever-se para a camada de saída:

$$\frac{\partial EQ}{\partial W'} = \frac{\partial EQ}{\partial y}.\frac{\partial y}{\partial W'} = \frac{\partial EQ}{\partial y}.\frac{\partial y}{\partial \beta}.\frac{\partial \beta}{\partial W'} \tag{2.27}$$

e para a camada escondida:

$$\frac{\partial EQ}{\partial W} = \frac{\partial EQ}{\partial y}.\frac{\partial y}{\partial W} = \frac{\partial EQ}{\partial y}.\frac{\partial y}{\partial \beta}.\frac{\partial \beta}{\partial W} = \frac{\partial EQ}{\partial y}.\frac{\partial y}{\partial \beta}.\frac{\partial \beta}{\partial y_h}.\frac{\partial y_h}{\partial W} \tag{2.28}$$

$$\frac{\partial EQ}{\partial W} = \frac{\partial EQ}{\partial y}.\frac{\partial y}{\partial \beta}.\frac{\partial \beta}{\partial y_h}.\frac{\partial y_h}{\partial \beta_h}.\frac{\partial \beta_h}{\partial W} \tag{2.29}$$

designando E=$(o - y)$, onde o é a saída pretendida e y é a saída obtida com a RN, podem calcular-se cada uma das derivadas:

$$\frac{\partial EQ}{\partial W'} = -2.E.\frac{\partial y}{\partial \beta}.y_h \tag{2.30}$$

com $\frac{\partial y}{\partial \beta}$ a depender da função de activação escolhida. Como foi referido, no caso da camada de saída é frequente a função de activação ser linear e será essa a situação

considerada aqui. Como a função de activação linear é constituída pela equação $y = \beta$, então $\frac{\partial y}{\partial \beta} = 1$ e pode escrever-se:

$$\frac{\partial EQ}{\partial W'} = -2.E.y_h \tag{2.31}$$

Para a camada escondida:

$$\frac{\partial EQ}{\partial W} = -2.E.\frac{\partial y}{\partial \beta}.W.\frac{\partial y_h}{\partial \beta_h}.\begin{bmatrix} I_l \\ 1 \end{bmatrix} \tag{2.32}$$

como $\frac{\partial y}{\partial \beta} = 1$:

$$\frac{\partial EQ}{\partial W} = -2.E.W.\frac{\partial y_h}{\partial \beta_h}.\begin{bmatrix} I_l \\ 1 \end{bmatrix} \tag{2.33}$$

onde $\frac{\partial y_h}{\partial \beta_h}$ representa a derivada da função de activação da segunda camada.

Com estas duas equações é possível fazer a actualização dos pesos da RN do modo a procurar que o IP baixe em direcção a um mínimo.

2.7.2 Optimização sem derivadas

Existem várias técnicas de optimização sem derivadas, sendo uma boa parte delas técnicas tradicionais de optimização que foram aplicadas à procura dos melhores valores para utilizar nos pesos das RNs. Dentro deste caso estão os Algoritmos Genéticos e o *Simulated Annealing*, tendo também sido utilizadas técnicas aleatórias [25].

Das soluções referidas apenas uma foi utilizada no decurso do presente trabalho, a que se refere aos algoritmos genéticos que serão descritos no capítulo 4, uma vez que a sua utilização não se resume apenas à optimização dos pesos.

2.7.3 Tipos de implementação dos algoritmos

Quase de forma independente do algoritmo a implementar existem formas distintas de utilizar o mesmo algoritmo. Algumas vezes as designações são algo confusas pelo que é conveniente esclarecer quais os tipos de implementação mais correntes.

Implementações *on-line* e *off-line*

As expressões *on-line* e *off-line* embora aparecendo por vezes associadas à implementação do algoritmo, não o estão efectivamente. Estas expressões referem-se à forma como o algoritmo é utilizado. Um algoritmo será designado *on-line* (em português é por vezes utilizada a expressão "em linha") se a identificação do modelo for efectuada directamente com os dados que estão a ser recolhidos do sistema e será designado *off-line* quando os dados são recolhidos na totalidade e só posteriormente são utilizados sem a necessidade da "presença física" do sistema que está a ser modelizado.

Implementações Recursivas e em Grupo

As implementações dos algoritmos podem ser em Grupo (do inglês *Batch*) ou Recursivas e esta opção condiciona a forma como o algoritmo é implementado na sua totalidade.

A implementação será designada em Grupo quando cada iteração de actualização dos pesos da RN é feita apenas após a avaliação de todo o conjunto de dados, ou seja após uma época de treino (época de treino, do inglês *training epoch* é a designação usada para o treino efectuado sobre todo o conjunto dos dados). Neste caso o resultado de cada par entrada/saída do conjunto de dados dá origem a uma contribuição para a actualização dos pesos da RN e no final da época é feita a média das contribuições para actualizar os pesos [4]. O objectivo deste procedimento é obter uma melhor aproximação das derivadas usadas no processo de actualização dos pesos.

A implementação será designada por Recursiva ou Iterativa se a actualização dos pesos da RN é feita após a avaliação de cada par entrada/saída dos dados de treino.

Estas duas situações estão representadas na figura 2.8, para permitir a compreensão mais rápida de ambas as formas de treino. Na figura, os pares de entrada/saída (com i entradas e j saídas) estão representados entre parêntesis rectos e o fluxo de treino é representado pelas setas.

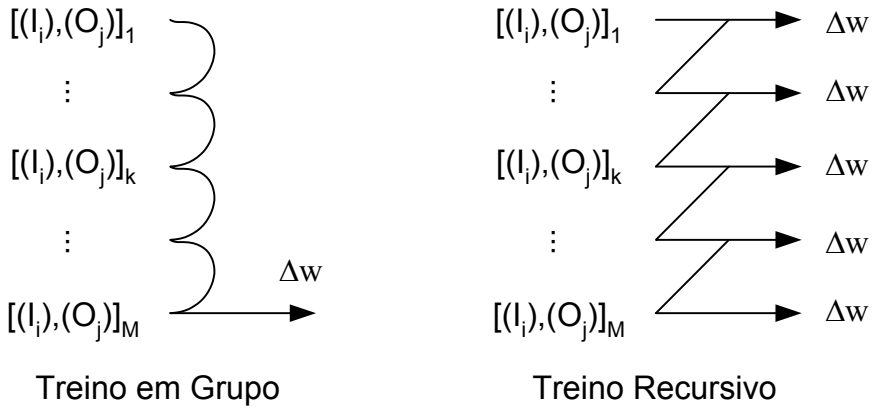


Figura 2.8: Ilustração dos processos de treino em grupo e do treino recursivo.

No presente trabalho foram utilizadas apenas implementações em grupo. Esta opção deve-se principalmente ao facto das implementações recursivas serem mais sensíveis à presença de ruído nos exemplos ou dados de treino.

Quando é apresentado à RN um exemplo que está afectado por ruído o seu efeito nos parâmetros da rede é imediato, fazendo os pesos afastarem-se do valor ideal, enquanto que no caso da implementação em grupo esse efeito é diluído nas contribuições do conjunto de dados de treino. O efeito imediato da amostra corrompida por ruído nos parâmetros da RN vai fazer com que os erros das amostras consecutivas sejam afectados pela actualização dos pesos da amostra corrompida enquanto no caso da versão em grupo este efeito é, sem dúvida, menor.

A utilização mais comum dos algoritmos é feita em grupo e utilizada *off-line*, embora potencialmente qualquer dos algoritmos apresentados possa ser implementado de ambas as formas. No entanto em alguns casos a implementação recursiva é mais difícil de obter.

Para um algoritmo como o *Steepest Descent* associado ao *Backpropagation* não existe uma dificuldade especial em fazer uma implementação recursiva, mas para os algoritmos que utilizam a matriz Hessiana ou as aproximações referidas nas secções anteriores, como o método de Gauss-Newton ou o Levenberg-Marquardt, para que o algoritmo seja processado rapidamente é evitado calcular a matriz e proceder depois à sua inversão usando-se normalmente um cálculo aproximado da matriz já invertida. Neste caso é particularmente difícil implementar o algoritmo Levenberg-Marquardt, uma vez que além de calcular a matriz Hessiana (ou a respectiva aproximação) é necessário adicionar uma matriz identidade e controlar o parâmetro μ por forma a garantir a convergência, ou seja é necessário garantir que a matriz é definida positiva e portanto invertível.

Existem, no entanto, aproximações ao cálculo da matriz Hessiana, como o algoritmo BFGS (Broyden-Fletcher-Goldfarb-Shanno) ou DFP (Davidon-Fletcher-Powell), habitualmente designados por métodos Quasi-Newton, que podem ser utilizados neste caso (ver, por exemplo, [26]).

Nas implementações recursivas para o método Gauss-Newton, para evitar problemas de convergência, torna-se mesmo necessário introduzir métodos como o Factor de Esquecimento ou o Traço Constante [1].

Em relação à aplicação dos algoritmos num processo *on-line* ou *off-line*, ambos os tipos de implementações podem ser usadas. Para implementações *on-line* é mais fácil a utilização de um algoritmo na forma Recursiva devido às restrições temporais decorrentes de efectuar o treino da RN e recolher os dados respeitando o período de amostragem. Para implementações *off-line*, o mais frequente é a utilização da implementação em grupo uma vez que, como anteriormente foi afirmado, são menos sensíveis às variações imediatas que podem ser provocadas, por exemplo, pela presença de ruído.

2.8 Estado da arte

2.8.1 Novos tipos de Redes Neuronais

Diversos novos paradigmas de RNs têm surgido e têm merecido maior ou menor atenção por parte da comunidade científica. Uma vez que o estudo destes tipos de RNs se encontra fora do âmbito proposto para este trabalho, refere-se apenas a título de exemplo um desses novos paradigmas, os *spiking neurons*, em torno dos quais existe já um largo trabalho, com desenvolvimento de algoritmos, exploração das suas capacidades (ver, por exemplo, [27]) e até desenvolvimento de *hardware* específico (ver, por exemplo, [28] e [29]).

2.8.2 Algoritmos de treino

Os algoritmos de treino para RNs constituem uma das áreas que continua a merecer atenção da comunidade científica, com particular relevância no que diz respeito ao desenvolvimento ou adaptação de algoritmos para funcionarem *on-line*, embora esta área não seja a única contemplada.

Em [30] é apresentado um algoritmo construído a partir do algoritmo de Gauss-Newton e que faz uso do facto do Jacobiano não ter característica máxima (do inglês *full rank*) para reduzir a complexidade dos cálculos necessários, separando os pesos absolutamente necessários dos supérfluos com base na característica. Para este algoritmo estão previstas duas versões: a versão simplificada que modifica apenas os pesos essenciais e uma completa que modifica todos os pesos com o objectivo de melhorar a convergência no caso de a característica ser demasiado baixa.

O artigo [31] apresenta, um algoritmo de treino genérico que pode ser utilizado para diversos tipos de estruturas usadas para modelização, aproveitando a característica comum de serem formados por um estágio não-linear seguido de um estágio linear. Esse algoritmo é retomado em [32], sendo também implementada uma versão *on-line* com base em janela deslizante (do inglês *sliding window*).

No artigo [33] e na tese [34] é apresentada uma adaptação do algoritmo de *Levenberg-Marquardt* por forma a permitir uma implementação recursiva. Este algoritmo modificado obtém-se pela alteração da equação de actualização dos pesos, que resulta de uma aproximação introduzida no cálculo da matriz Hessiana e facilita a sua inversão e por uma nova forma de avaliar a vizinhança que determina a validade da presente iteração, da qual resulta uma forma distinta de calcular o parâmetro μ_k.

Em [35] é apresentada uma variante do algoritmo de *Levenberg-Marquardt* (designado *Neighborhood Based Levenberg-Marquardt Algorithm for Neural Network Training*) que consiste basicamente em treinar partes da RN de cada vez (daqui resulta a designação de vizinhanças - do inglês *neighborhood*). As vizinhanças da RN a treinar têm sempre que incluir saídas e entradas, como de resto é fácil de concluir tendo em conta o algoritmo e é permitida a sobreposição entre vizinhanças. O objectivo desta variante é diminuir a complexidade computacional da implementação quer em termos de memória necessária quer no tempo de treino de uma RN. A divisão da rede em vizinhanças facilmente concretiza estes objectivos (salvaguardando que não haja demasiada sobreposição das vizinhanças) uma vez que, como é explicado, sendo m o número de pesos da RN é necessário calcular e armazenar m^2 elementos e executar aproximadamente m^3 operações. Pelos resultados apresentados estes objectivos são concretizados mas à custa de uma performance ligeiramente inferior do que a obtida para o algoritmo de *Levenberg-Marquardt* completo.

Uma modificação de um algoritmo de treino (*steepest descent*) por um motivo diferente é apresentada em [36]. Os autores verificaram que o facto de existirem pesos de valor absoluto mais elevado é prejudicial para a tolerância a falhas parciais, uma vez que se a falha atingir esta ligação associada ao peso de maior valor absoluto, existirá um maior impacto na saída da RN. O novo algoritmo é muito semelhante ao *steepest descent*, sendo a principal modificação associada ao facto de as alterações aos pesos

que excedem a média dos valores absolutos dos pesos já existentes serem rejeitadas.

2.8.3 Outros tópicos

Alguns outros tópicos têm merecido também a atenção dos investigadores da área.

O artigo [37] apresenta um resumo dos resultados teóricos sobre complexidade de RNs que abrange as redes do tipo perceptrão, RBF, *winner-take-all* e *spiking neurons*. Os resultados apresentados não incluem os aspectos de aprendizagem da RN.

2.9 Conclusão

Neste capítulo após uma curta introdução sobre RNs, com maior ênfase nas FNNs que foram escolhidas pela sua maior divulgação ao nível das ferramentas desenvolvidas de livre utilização, foram apresentados os algoritmos de treino mais comuns aplicáveis a estas últimas: *Backpropagation* com *Steepest Descent*, Newton, Gauss-Newton e Levenberg-Marquardt. Foram também discutidas as formas de implementação desses algoritmos, nomeadamente as implementações *on-line*, *off-line*, Recursivas e em Grupo e analisado, de forma breve, o estado da arte.

Capítulo 3

Identificação de sistemas com Redes Neuronais

"A model of a system is a description of (some of) its properties, suitable for a certain purpose. The model need not be a true and accurate description of the system, nor need the user have to believe so, in order to serve its purpose." - Lennart Ljung, em System Identification - Theory for the User, cientista. (1946 -)

3.1 Introdução

A identificação de sistemas é a área científica relativa à construção ou selecção de modelos de sistemas dinâmicos para servirem determinados propósitos [38].

Uma vez analisadas as estruturas das RNs, em especial as FNNs, e os respectivos algoritmos de treino é necessário saber como se procede à identificação de um modelo, directo ou inverso, de forma a poder ser usado numa estrutura de controlo.

Este capítulo está dedicado à identificação de sistemas com redes neuronais, sendo que a maioria dos resultados já conhecidos da teoria dos sistemas lineares se aplica directamente, ainda que com um ligeiro aumento de complexidade [1].

O processo de identificação necessita de diversos passos, começando habitualmente pela aquisição dos dados, seguindo-se a determinação da ordem do sistema ou do número de regressores, pré-processamento dos dados, escolha da classe de modelos a utilizar, utilização de uma estrutura de treino e verificação da qualidade do modelo. A figura 3.1 apresentada em [1], representa, de forma simplificada, o procedimento que leva à identificação de um sistema e à consequente criação de um modelo. Vale a pena notar que este processo pode ser iterativo caso o(s) primeiro(s) modelo(s) não sejam satisfatórios. A repetição do processo de identificação não implica necessariamente a repetição de todos os passos uma vez que se pode verificar que, apesar de o processo não ter sido bem sucedido, algumas das fases iniciais foram correctamente concluídas.

A fase inicial designada por aquisição de dados diz respeito à obtenção dos dados e à sua preparação para criar modelos.

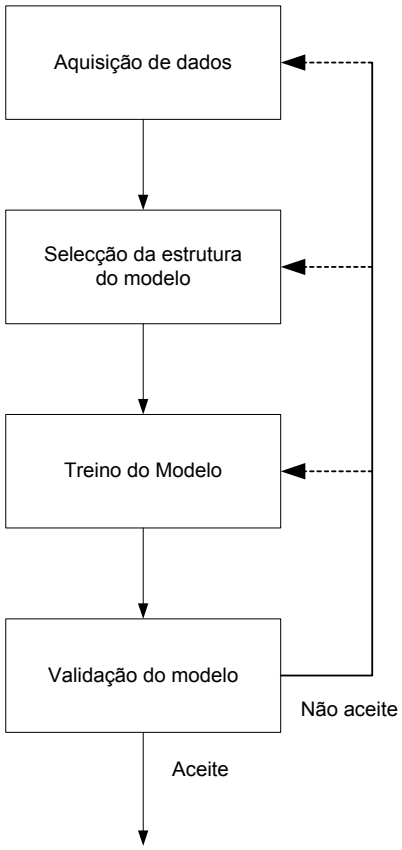


Figura 3.1: Procedimento de identificação de sistemas.

A selecção da estrutura do modelo é a fase em que se escolhe a classe de modelo que se pretende usar e a sua dimensão. Uma grande parte das classes de modelos que são usados com RNs resultam das classes de modelos lineares, sendo depois transpostos para as RNs, através da utilização dos mesmos regressores [1].

A validação dos modelos corresponde a verificar se o modelo produzido serve a finalidade a que se destina. No caso concreto dos modelos baseados em RNs é necessário verificar a capacidade de generalizar da RN utilizada, ou seja, garantir que a RN se comporta da forma pretendida não só no caso dos exemplos com que foi treinada, mas também na presença de situações distintas.

Este capítulo introduz também a primeira contribuição original desta tese: a Técnica Híbrida Genérica/Especializada para modelos Inversos, que combina treino e avaliação da qualidade de modelos inversos.

3.2 Aquisição de dados

A aquisição de dados, apesar de poder à primeira vista parecer uma operação demasiado simples para constituir uma fase do processo de identificação é, efectivamente, uma fase

uma vez que só se pode dizer que está concluída quando a identificação do sistema é bem sucedida. Esta etapa comporta a escolha das condições experimentais, do período de amostragem e a escolha do tipo de dados a usar.

3.2.1 Condições experimentais

No sentido de se conseguir obter dados de boa qualidade, é necessário garantir que as condições experimentais são favoráveis, ou seja, condições de baixo ruído e ausência de perturbações significativas.

3.2.2 Escolha do período de amostragem e estudo preliminar do sistema

Para escolher o período de amostragem e efectuar um estudo preliminar do sistema são normalmente efectuadas diversas experiências com o objectivo de recolher informação. Em concreto, pretende-se, identificar o grau de não-linearidade e estimar o tempo de subida para determinar o período de amostragem correcto.

Sem tentar incorrer em generalizações, pode-se afirmar que, em processamento digital de sinal, o intervalo de amostragem deve ser o mais pequeno que a implementação prática permitir [39]. No entanto a situação no projecto de sistemas de controlo digital é diferente.

No caso dos sistemas de controlo em malha fechada, uma escolha racional do intervalo de amostragem deve ser baseada no conhecimento da sua influência para o desempenho final do sistema. É razoável admitir que a máxima frequência de interesse deve estar relacionada com a largura de banda do sistema em malha fechada. Assim, a escolha do intervalo de amostragem pode basear-se na largura de banda do sistema ou, o que é equivalente, no seu tempo de subida [39].

De acordo com [40], existe uma forma prática de estabelecer uma relação entre o tempo de subida e o período de amostragem para sistemas de primeira ordem, sendo a relação dada por:

$$N_r = \frac{T_r}{h} \simeq 4 \ a \ 10 \tag{3.1}$$

onde T_r é o tempo de subida, h é o período de amostragem e N_r é o número de amostras que o tempo de subida deve conter, ou seja, o tempo de subida deverá ser 4 a 10 vezes o período de amostragem. Embora sejam valores para sistemas de primeira ordem e nesta fase a ordem do sistema seja ainda desconhecida, esta aproximação é frequentemente usada. Em [40] é também indicada a forma de obter valores para sistemas de segunda ordem, mas neste caso não se trata de uma regra prática tão simples uma vez que é necessário conhecer características adicionais do sistema.

A regra prática da equação 3.1 pode ser aplicada em todos os casos, embora o seu suporte teórico seja vago. A restante literatura propõe soluções para a escolha do período de amostragem que são, em alguns casos mais difíceis de aplicar.

Em [38] é sugerido que a frequência de amostragem seja cerca de dez vezes superior à largura de banda do sistema.

Em [41] é indicado que é prática comum escolher o período de amostragem de acordo com:

$$w_B h \approx 0,5 \ a \ 1 \tag{3.2}$$

onde w_B é a frequência para a qual o ganho de malha fechada baixa para 0,7 e h o período de amostragem.

Vale a pena referir dois exemplos de controlo publicados nas páginas de *internet* das universidades do Michigan [42] e de Guelph [43], utilizados para aulas de demonstração, em que são escolhidos valores relativos ao número de amostras por tempo de subida muito superiores aos propostos pela equação 3.1.

Nesta fase é também habitual realizarem-se testes para analisar a linearidade do sistema por forma a saber que tipo de controlo se deverá aplicar. Se o sistema for linear não se justifica fazer uma identificação não-linear e obviamente as opções mais simples deverão ser sempre consideradas [1].

3.2.3 Escolha do tipo de dados a utilizar

Na fase de aquisição de dados para posterior identificação de modelos do sistema, uma das preocupações é escolher o sinal de entrada adequado.

Para sistemas não-lineares, é necessário escolher não só um espectro de frequência alargado como também uma gama de amplitudes abrangente [15]. Por outro lado é necessário decidir entre usar um sinal de malha aberta ou de malha fechada.

Normalmente é usado sinal de malha aberta, embora em algumas situações sinal de malha fechada possa ser preferido uma vez que o sistema é mantido dentro da gama de valores onde é suposto operar [1], mas esta segunda solução traz o problema da correlação entre sinal de entrada e de saída que pode levar à perda de capacidade de identificação do sistema [44].

3.3 Preparação dos dados

Após a fase de aquisição de dados é necessário prepará-los para fazer o treino dos modelos. Para tal é normalmente necessário escalar os dados, fazer o pré-processamento, filtragem e separação dos mesmos em diversos conjuntos que se destinam a funções distintas, como será explicitado.

3.3.1 Escalar os dados

Como é referido em [38] é fortemente recomendada a aplicação de um factor de escala aos dados, podendo ser para tal apresentadas as seguintes justificações:

- é provável que os sinais sejam medidos em unidades físicas diferentes e haverá uma tendência para que o sinal de maior magnitude domine o modelo [15].

- o algoritmo de treino torna-se mais robusto e leva a uma convergência mais rápida [45].

- as funções de activação têm na sua maioria um comportamento constante a partir de determinados valores da entrada, fazendo com que a resposta da função de activação possa ser idêntica para valores muito distintos da entrada.

- a experiência mostrou que foram obtidos modelos mais precisos [1].

A aplicação de um factor de escala aos dados significa subtrair aos sinais a sua média e dividi-los pelo seu desvio padrão como é referido em [15]. Esta situação está ilustrada na figura 3.2, onde os sinais com índice s representam sinais escalados (do inglês *scaled*).

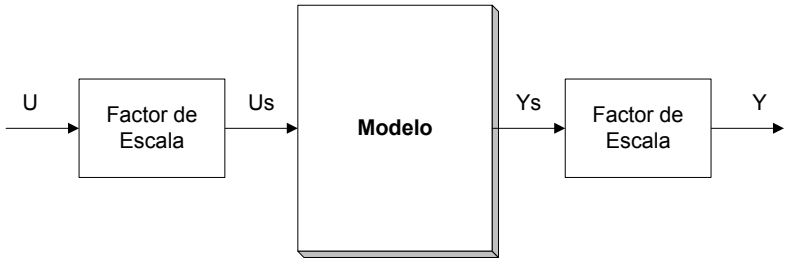


Figura 3.2: Aplicação do factor de escala aos modelos.

Nem sempre a aplicação do factor de escala é feita imediatamente antes e depois do modelo, uma vez que dentro de um sistema mais complexo (por exemplo uma malha de simulação de controlo) podem todos os sinais ser escalados e só quando são necessários no exterior se proceder à aplicação do factor de escala inverso. Esta questão pode levantar alguns problemas no caso de malhas mais complexas, uma vez que alguns blocos poderão trabalhar com sinais afectados por um factor de escala e outros não. Este assunto será retomado no decurso desta tese.

Embora o processo aqui explicado de aplicação de um factor de escala aos sinais seja bastante simples, nem sempre a sua aplicação se revela tão trivial. Como aplicar um factor de escala a sinais cuja média e variância não sejam ainda conhecidas?

Por exemplo, na utilização de um modelo neuronal num sistema de controlo de uma aplicação real, o sinal de controlo e o sinal controlado não são conhecidos antes da aplicação do sinal de controlo, pelo que a sua média e variância não são conhecidas. No entanto, o modelo neuronal necessita trabalhar com sinais escalados.

Poderá argumentar-se que a utilização da média e variância do sinal de treino são uma solução viável, e de facto são, embora com algumas limitações, mas como proceder num processo de treino *on-line* no qual não existe uma média e variância provenientes do sinal de treino? Estando a decorrer o próprio processo de recolha de dados, se os valores de média e variância forem repetidamente calculados obter-se-ão valores diferentes que terão impacto no treino da RN, podendo torná-lo instável.

Em ambos os casos, os problemas enunciados apontam para uma fixação dos valores da média e da variância, mas naturalmente estes valores fixados à priori serão diferentes dos valores reais. Que impacto tem esta limitação no desempenho dos modelos implementados com RNs?

Se os modelos forem de boa qualidade, desde que a média e variância "abranjam" a totalidade da zona de funcionamento do modelo, não deverão existir problemas. Caso contrário os sinais ao sofrerem a aplicação de um factor de escala poderão ser colocados indevidamente na zona de saturação da função de activação e levar a resultados inesperados.

3.3.2 Pré-processamento dos dados

Em diversas aplicações de RNs é feito pré-processamento dos dados, eliminando dados redundantes e *outliers* [46] [38].

Eliminação de dados redundantes

Consideram-se dados redundantes aqueles que representam o mesmo regime de funcionamento de um determinado sistema, não contribuindo por isso para a melhor caracterização do sistema.

A eliminação de dados redundantes é conveniente uma vez que para além de se traduzirem num acréscimo de tempo de treino, tornarão o modelo mais apto em alguns regimes de funcionamento (aqueles que estão mais representados) e menos apto noutros regimes [1].

Na modelização de certos sistemas a eliminação de dados redundantes pode ser complexa de efectuar devido à dependência que a saída tem das amostras anteriores. Em sistemas dinâmicos ou com história, os modelos não poderão ser treinados com dados onde se perca a sequência das amostras anteriores.

Remoção de *outliers*

Um *outlier* é uma amostra cujo valor de saída não corresponde correctamente à entrada. Esta situação pode ocorrer por diversos motivos, sendo os mais comuns uma falha de leitura e a influência de ruído. Para se identificar um *outlier* é necessário ter um conhecimento já alargado do sistema a modelizar.

Os *outliers* podem ser removidos ou as amostras correspondentes substituídas por valores estimados. Sempre que possível a identificação dos *outliers* deve ser feita durante a fase de aquisição de dados por forma a permitir a leitura de um novo valor.

3.3.3 Filtragem

A filtragem é largamente usada para remover ruído, perturbações periódicas, componentes contínuas e efeitos dinâmicos que não sejam interessantes para a modelização

do sistema [1]. Filtrar os dados, antes de criar os modelos, utilizando filtros que melhorem a relação sinal/ruído nas bandas de interesse do sistema, melhora a qualidade da modelização nessas bandas [47] [38].

Em [48] é sugerido que podem ser usados filtros *anti-alising* analógicos e filtros digitais para a filtragem após amostragem.

3.3.4 Divisão dos dados em diversos conjuntos

Os dados recolhidos são habitualmente divididos em diversos conjuntos que serão utilizados em diferentes funções. O mais frequente é dividir os dados em dois conjuntos, sendo destinado um para a fase de treino e outro para a de teste ou validação do modelo. A necessidade de fazer esta divisão resulta do problema de *overtraining*, uma vez que a RN pode aprender informação específica do conjunto de dados (como o ruído e as características próprias do sinal). A uma situação de *overtraining* diz-se que corresponde uma RN que faz *overfitting*, ou seja, que está demasiado adaptada ao conjunto de dados de treino. Esta situação está explicada na secção referente à capacidade de generalizar. Portanto a sequência de treino é utilizada para fazer a aprendizagem da RN e a de teste é utilizada após o treino (total ou parcial) para verificar a capacidade de generalizar da RN.

Ainda que menos comum, também é possível encontrar na literatura autores que usam três conjuntos de dados. Um exemplo de divisão em três conjuntos de dados pode ser encontrado no epílogo de [4] e em [49] onde é proposto usar um sequência de dados para treinar a RN, uma segunda sequência para validar a capacidade de generalizar a RN e uma terceira para comparar a qualidade de diferentes RNs.

3.4 Classes de modelos

Antes de ser possível fazer um modelo é necessário escolher o tipo ou classe de modelo que se pretende usar. Uma grande parte das classes de modelos que são usados com RNs resultam das classes de modelos lineares, sendo depois transpostos para a utilização com RNs, através da utilização dos mesmos regressores [1].

A apresentação que se segue das classes de modelos lineares é semelhante à feita por Ljung [38] e por Magnus Nørgaard [1], sendo apenas apresentados modelos discretos no tempo.

3.4.1 Classes de modelos lineares

Um modelo pode ser classificado como linear [38] se puder ser descrito de acordo com a equação 3.3.

$$y(k) = G(q^{-1}).u(k) + H(q^{-1}).e(k) \tag{3.3}$$

sendo q^{-1} o operador atraso e k o número da amostra (correspondente ao instante de tempo kh, sendo h o período de amostragem).

Uma estrutura mais genérica de um modelo está representada na equação 3.4. Esta estrutura de modelo é designada por genérica uma vez que a partir dela podem ser obtidas as diferentes classes através de simplificações sobre os polinómios que a compõem.

$$A(q^{-1})y(k) = q^{-td}.\frac{B(q^{-1})}{F(q^{-1})}.u(k) + \frac{C(q^{-1})}{D(q^{-1})}.e(k) \tag{3.4}$$

Na equação 3.4 y e u são, respectivamente, os sinais de saída e de entrada do sistema, o sinal e representa ruído branco, td representa a aproximação inteira do valor do tempo morto e os polinómios são:

$$\begin{aligned}
A(q^{-1}) &= 1 + a_1.q^{-1} + ... + a_n.q^{-n} && (3.5)\\
B(q^{-1}) &= b_0 + b_1.q^{-1} + ... + b_m.q^{-m} && (3.6)\\
C(q^{-1}) &= 1 + c_1.q^{-1} + ... + c_i.q^{-i} && (3.7)\\
D(q^{-1}) &= 1 + d_1.q^{-1} + ... + d_j.q^{-j} && (3.8)\\
F(q^{-1}) &= 1 + f_1.q^{-1} + ... + f_r.q^{-r} && (3.9)
\end{aligned}$$

onde n, m, i, j e r representam, respectivamente a ordem dos polinómios A, B, C, D e F.

A partir da equação 3.3 pode escrever-se, sem perda de generalidade, a equação de predição de uma amostra (do inglês *one step ahead prediction*), sendo $\hat{y}(k+1)$ a melhor predição possível do sinal $y(k)$, conhecidas as amostra até $y(k-1)$:

$$\hat{y}(k+1) = H^{-1}(q^{-1}).G(q^{-1}).u(k) + (1 - H(q^{-1})).y(k) \tag{3.10}$$

A equação de predição é fundamental na modelização com RNs, uma vez que se pretende implementar modelos dos sistemas em estudo, o que corresponde a implementar estas equações.

Estrutura *Finite Impulse Response* (FIR)

A estrutura FIR é a mais simples que é possível obter a partir do modelo genérico e corresponde às seguintes simplificações nos polinómios:

$$G(q^{-1}) = q^{-td}.B(q^{-1}) \tag{3.11}$$

$$H(q^{-1}) = 1 \tag{3.12}$$

Neste caso a equação de predição será:

$$\hat{y}(k+1) = q^{-td}.B(q^{-1}).u(k) \tag{3.13}$$

Esta equação pode também ser expressa na forma de equação de regressão:

$$\hat{y}(k+1) = \theta^T \varphi(k) \tag{3.14}$$

onde θ representa o vector de parâmetros ajustáveis no modelo e φ representa o vector de regressão que, neste caso, será composto por:

$$\varphi(k) = [u(k - td), ..., u(k - td - m)] \tag{3.15}$$

e θ será:

$$\theta = [b_0, ..., b_m] \tag{3.16}$$

Um modelo desta classe tem algumas limitações, nomeadamente do ponto de vista prático. Como se pode ver pela equação 3.13, o modelo não inclui nenhuma informação sobre a saída e, sendo de ordem finita, não pode descrever de forma exacta um sistema com pólos. Normalmente dá origem a um número muito elevado de parâmetros.

Estrutura *AutoRegressive with eXogenous signal* (ARX)

As simplificações introduzidas no modelo, neste caso, são:

$$G(q^{-1}) = \frac{q^{-td}.B(q^{-1})}{A(q^{-1})} \tag{3.17}$$

$$H(q^{-1}) = \frac{1}{A(q^{-1})} \tag{3.18}$$

A equação de predição será:

$$\hat{y}(k+1) = q^{-td}.B(q^{-1}).u(k) + (1 - A(q^{-1})).y(k) \tag{3.19}$$

A equação pode também ser expressa na forma de equação de regressão:

$$\hat{y}(k+1) = \theta^T.\varphi(k) \tag{3.20}$$

sendo φ :

$$\varphi(k) = [y(k), ..., y(k - n), u(k - td), ..., u(k - td - m)] \tag{3.21}$$

e θ será:

$$\theta = [-a_1, ..., -a_n, b_{0,}..., b_m] \tag{3.22}$$

Estrutura *AutoRegressive Moving Average with eXogenous signal* (ARMAX)

Esta estrutura corresponde a fazer as seguintes simplificações no modelo:

$$G(q^{-1}) = \frac{q^{-td}.B(q^{-1})}{A(q^{-1})} \tag{3.23}$$

$$H(q^{-1}) = \frac{C(q^{-1})}{A(q^{-1})} \tag{3.24}$$

A equação de predição será:

$$\hat{y}(k+1) = q^{-td}.B(q^{-1}).u(k) + (1 - A(q^{-1})).y(k) + (C(q^{-1}) - 1).\epsilon(k) \tag{3.25}$$

onde $\epsilon(k) = y(k) - \hat{y}(k)$ representa o erro de predição.

A equação pode também ser expressa na forma de equação de regressão:

$$\hat{y}(k+1) = \theta^T.\varphi(k) \tag{3.26}$$

sendo φ :

$$\varphi(k) = [y(k), ..., y(k-n), u(k-td), ..., u(k-td-m), \epsilon(k-1), ..., \epsilon(k-i)] \tag{3.27}$$

e θ será:

$$\theta = [-a_1, ..., -a_n, b_0, ..., b_m, c_1, ..., c_i] \tag{3.28}$$

Deve notar-se que neste caso o sinal ϵ representa a diferença entre a saída estimada e a saída pretendida, o que faz depender o vector de regressão dos parâmetros do modelo. Esta situação faz com que o processo de treino dos modelos se torne mais complexo.

Output Error model structure

As simplificações introduzidas no modelo neste caso são:

$$G(q^{-1}) = \frac{q^{-td}.B(q^{-1})}{F(q^{-1})} \tag{3.29}$$

$$H(q^{-1}) = 1 \tag{3.30}$$

A equação de predição será:

$$\hat{y}(k+1) = q^{-td}.B(q^{-1}).u(k) + (1 - F(q^{-1})).\hat{y}(k) \tag{3.31}$$

Esta equação pode também ser expressa na forma de equação de regressão:

$$\hat{y}(k+1) = \theta^T.\varphi(k) \tag{3.32}$$

sendo φ :

$$\varphi(k) = [\hat{y}(k), ..., \hat{y}(k-r), u(k-td), ..., u(k-td-m)] \tag{3.33}$$

e θ será:

$$\theta = [-f_1, ..., -f_r, b_0, ..., b_m] \tag{3.34}$$

Neste caso o modelo tem um vector de regressão que inclui valores estimados, ou seja, o próprio vector de regressão depende dos parâmetros do modelo. Esta situação torna a identificação mais complexa [1].

3.4.2 Classes de modelos não-lineares baseados em Redes Neuronais

As classes de modelos não-lineares baseados em RNs mais utilizadas são extensões directas das correspondentes classes lineares, usando assim as mesmas entradas e sendo a estrutura interna composta por RNs como foi proposto por vários autores [50] [15] [51] [1].

As classes serão designadas pelos nomes correspondentes precedidos pela sigla NN (do inglês *Neural Network*). Por exemplo será designado por NNARX um modelo cujas entradas são as mesmas que o correspondente modelo ARX, ou seja valores passados dos sinais de entrada e de saída. A figura 3.3 apresenta o diagrama de blocos de um modelo desta classe.

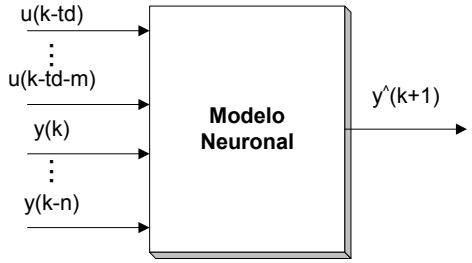


Figura 3.3: Exemplo de um modelo da classe NNARX.

Da mesma forma um modelo NNARMAX está ilustrado na figura 3.4.

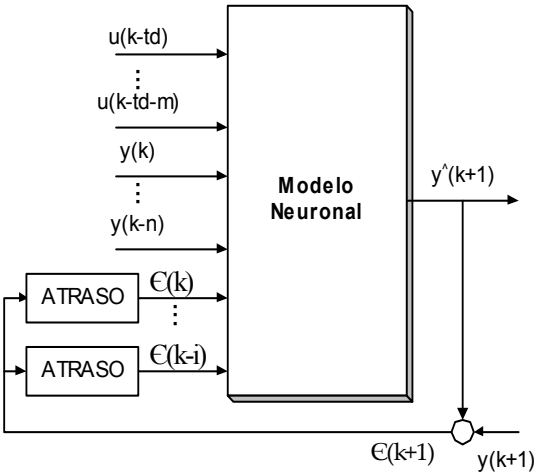


Figura 3.4: Exemplo de um modelo da classe NNARMAX.

Um modelo NNFIR está ilustrado na figura 3.5.

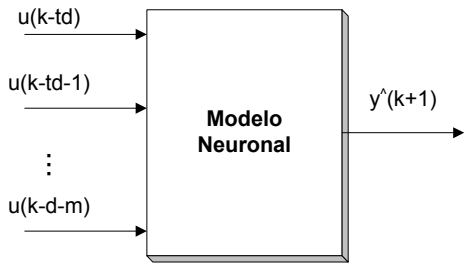


Figura 3.5: Exemplo de um modelo da classe NNFIR.

Um modelo NNOE está ilustrado na figura 3.6.

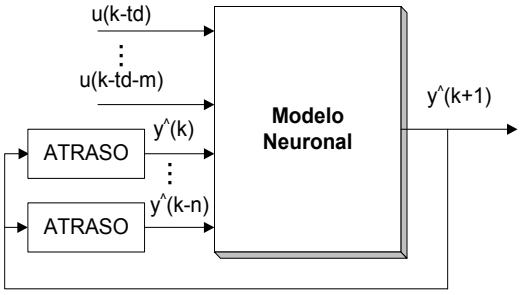


Figura 3.6: Exemplo de um modelo da classe NNOE.

Os tipos de modelos não-lineares baseados em RNs merecem, no entanto, uma análise mais aprofundada no sentido de verificar a sua facilidade de utilização.

As classes NNFIR e NNARX têm a vantagem de ser sempre estáveis, dado que existe uma relação algébrica pura entre a predição e os regressores [1], o que é particularmente importante para os sistemas não-lineares, fazendo com que sejam estas as alternativas preferenciais, principalmente a classe NNARX uma vez que a NNFIR, à semelhança da FIR, ao não incluir nenhuma informação sobre as saídas anteriores, não pode descrever correctamente um sistema com pólos.

A classe NNARMAX, como se pode ver pela equação 3.27, tem no seu conjunto de regressores o erro entre a saída obtida pelo modelo e a saída pretendida. Este facto introduz uma realimentação no próprio modelo o que pode ter implicações na sua estabilidade.

A classe NNOE tem uma situação semelhante por incluir no seu conjunto de regressores a predição das saída anteriores como pode ser verificado na equação 3.33.

3.5 Ordem do sistema

Na identificação de sistemas "clássica" para se construir um modelo do sistema é essencial determinar a sua ordem, ou seja, determinar qual a ordem das equações necessária

para caracterizar o sistema com determinado grau de exactidão ou encontrar um ponto de equilíbrio entre facilidade de estabelecer o modelo e a sua exactidão.

Na modelização com RNs a situação é semelhante, pois ao aumento do número de regressores do sistema corresponde um rápido aumento dos parâmetros. No caso das RNs a expressão "ordem" deve ser substituída pela expressão "número de regressores", uma vez que o conceito de ordem não encontra aplicação directa nos modelos neuronais.

Sendo NCE o número de neurónios na camada escondida, NE o número de entradas e NS o número de saídas, o número de parâmetros, NP, é dado por:

$$NP = NCE.(NE + 1) + NS.NCE + 1 \tag{3.35}$$

Portanto ao aumento da ordem utilizada para identificar o sistema corresponde uma maior dificuldade em fazer o treino da RN devido ao maior número de parâmetros.

Para a determinação da ordem do sistema pode ser usada uma função de custo semelhante à da equação 2.3 ou 2.4 e efectuado um teste do valor da função de custo *versus* a ordem/número de regressores do modelo [39].

Considerando um sistema real para o qual existe um modelo Λ de ordem σ capaz de descrever o seu comportamento dinâmico, supondo que o número de amostras é elevado e que o sistema não está afectado por ruído, deve ser possível obter um modelo Λ_x, capaz de descrever o comportamento do sistema, desde que a sua ordem seja igual ou superior à do modelo Λ. Neste caso ideal a função de custo deverá ser constante e igual a zero. A figura 3.7 representa uma hipotética evolução do valor da função de custo em função da ordem do modelo/número de regressores.

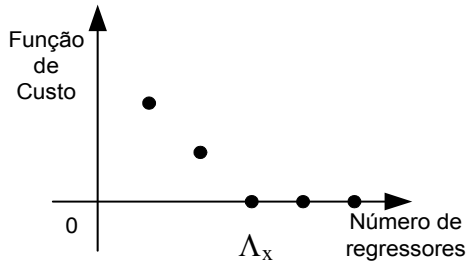


Figura 3.7: Exemplo da evolução da função de custo em função do número de regressores do modelo.

Este comportamento será ligeiramente distinto se o sistema estiver afectado por ruído. A figura 3.8 representa uma hipotética evolução do valor da função de custo em função da ordem do modelo/número de regressores para um sistema afectado por ruído.

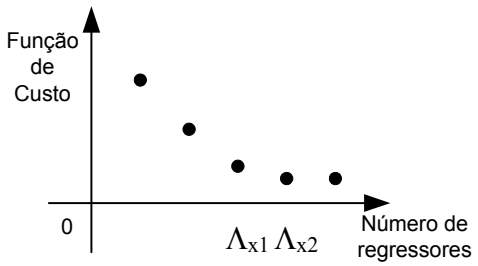


Figura 3.8: Exemplo da evolução da função de custo em função do número de regressores do modelo para um sistema afectado por ruído.

Da análise da figura pode-se concluir que a função de custo decresce com o número de regressores e que esta diminuição é lenta a partir do modelo cuja ordem é "igual" à ordem do sistema. Subsiste deste modo, o problema de decidir quando se considera que a função de custo é "suficientemente baixa" para indiciar um modelo de complexidade adequada [44].

3.6 Estruturas de treino

Como já foi visto, para treinar uma RN é necessário um algoritmo de treino, mas existem diversas formas de utilizar os algoritmos propostos na literatura. Essas formas correspondem às estruturas de treino que serão descritas nos pontos seguintes.

Em geral, pretende-se minimizar uma função de custo semelhante às equações 2.3 ou 2.4, que neste capítulo será escrita de forma a tornar explícita a dependência com os parâmetros do modelo θ e com a sequência de treino Z^N, que contém N pontos, como pode ser visto na equação 3.36.

$$V_N(\theta, Z^N) = \frac{1}{N}.\sum_{i=1}^{N}(o - \hat{o})^2 \tag{3.36}$$

Nas diferentes estruturas de treino são minimizadas funções de custo semelhantes à da equação 3.36, variando apenas a saída o e a saída $\hat{o}$ estimada pelo modelo.

Para o treino ser bem sucedido, é necessário escolher o sinal $u(k)$ apropriado de acordo com a secção 3.2.3.

3.6.1 Modelo directo

A estrutura clássica de treino para modelos directos está representada na figura 3.9. O procedimento de treino consiste em colocar o modelo neuronal em paralelo com o sistema que se pretende modelizar. O erro resultante da diferença entre o sistema e o modelo é usado para corrigir os pesos da RN usando o algoritmo escolhido.

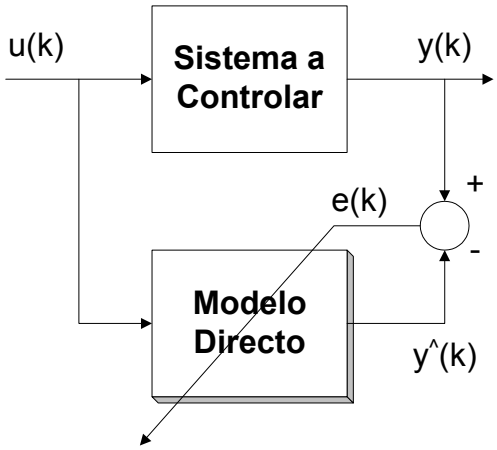


Figura 3.9: Diagrama de blocos da estrutura de treino do modelo directo.

3.6.2 Modelo inverso

Apesar da diversidade de formas de treino propostas na literatura podem definir-se dois importantes grupos: os modelos genéricos (correspondentes ao *general training*) e os modelos especializados (correspondentes ao *specialized training* [7]).

Para um sistema genérico que possa ser descrito pela equação:

$$y(k+1) = g[y(k), ..., y(k-n+1), u(k), ..., u(k-m+1)] \tag{3.37}$$

O modelo inverso pode ser obtido da seguinte forma:

$$\hat{u}(k) = g^{-1}[y(k+1), ..., y(k-n+1), u(k-1), ..., u(k-m+1)] \tag{3.38}$$

onde ^indica que o parâmetro, ou a função em causa, é estimado.

A equação 3.38 será sempre implementada quando se pretende um modelo inverso, variando a estrutura usada durante o treino.

Modelo inverso genérico

A estrutura usada para o modelo inverso genérico surge de forma "natural" pela sua proximidade com a usada para treinar os modelos directos como pode ser visto na figura 3.10.

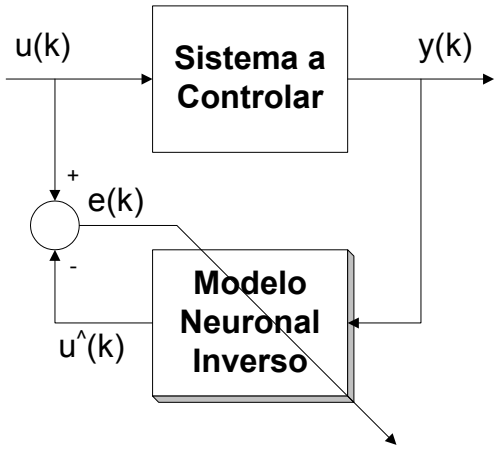


Figura 3.10: Diagrama de blocos da estrutura de treino usada para criar modelos inversos genéricos.

Da figura 3.10 pode concluir-se que a função de custo a minimizar durante o treino da rede neuronal será do tipo:

$$V_N(\theta, Z^N) = \frac{1}{N}.\sum_{k=1}^{N}(u(k) - \hat{u}(k))^2 \tag{3.39}$$

o que constitui uma das desvantagens deste tipo de estrutura de controlo, como adiante será explicitado. Em geral pretende-se minimizar uma função do tipo:

$$V_N(\theta, Z^N) = \frac{1}{N}.\sum_{k=1}^{N}(r(k) - y(k))^2 \tag{3.40}$$

uma vez que o objectivo final é que o erro na saída seja minimizado.

Esta é a principal desvantagem que se prende com o facto de o treino não ser feito numa situação semelhante à situação real de controlo e ser difícil garantir que a gama dinâmica de $u(k)$ é razoavelmente coberta por forma a que exista uma certeza de que a rede neuronal vai ser capaz de responder correctamente a entradas diferentes daquelas com que foi treinada [1], [52]. Esta situação é frequentemente designada na literatura por um treino que não é dirigido ao objectivo (do inglês *goal directed*). Outra desvantagem apontada a esta solução é a possibilidade de, se o sistema a modelizar não for bi-unívoco, se obter um modelo inverso incorrecto.

É possível tentar abranger toda a gama dinâmica de $u(k)$ de forma bem representativa, mas frequentemente o que é conhecido é a zona de funcionamento do sistema e não os valores de $u(k)$ que levam a essa zona pelo que é difícil garantir que vamos obter um modelo inverso de boa qualidade para ser usado como controlador [53]. Da desvantagem anterior resulta que se o sistema a controlar não é bi-unívoco pode obter-se um modelo inverso incorrecto, uma vez que nem sequer é possível garantir que estamos a trabalhar com sinais idênticos aos da malha de controlo [52], [7], [23] e [15].

Modelo inverso especializado

Para resolver os problemas referidos para a estrutura de treino do modelo inverso genérico surgiu a estrutura do modelo inverso especializado que pode ser observada na figura 3.11[1].

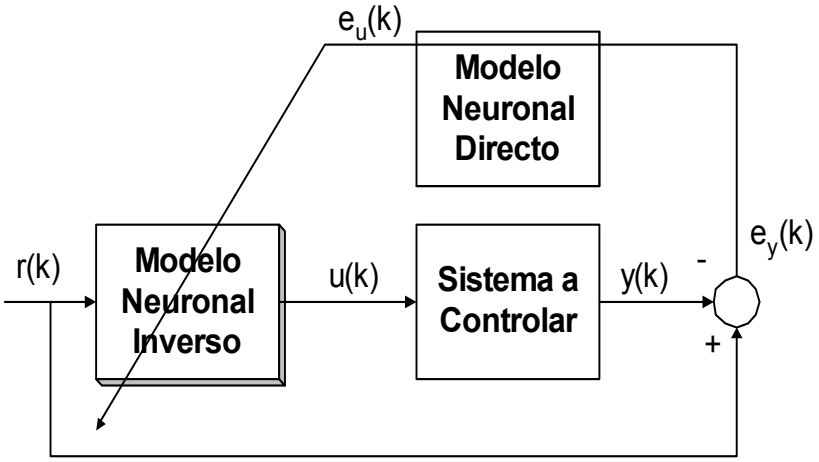


Figura 3.11: Diagrama de blocos da estrutura de treino de um modelo inverso especializado.

Nos modelos inversos especializados o critério a minimizar é o da equação 3.40, como se pode concluir facilmente do diagrama de blocos da figura 3.11, que representa este tipo de treino, onde $e_y(k)$ é o erro relativo à saída $y(k)$ e $e_u(k)$ é o erro relativo ao sinal de controlo.

Analisando este diagrama de blocos podemos concluir que para evitar a desvantagem de não ser *goal directed*, o modelo inverso é treinado na mesma situação em que estará dentro da malha de controlo. Pode obter-se nesta situação um erro relativo à saída que não pode ser aplicado directamente para o treino do modelo inverso, sendo necessário relacionar este erro com o erro na saída do bloco que está a ser optimizado. Ou seja, o erro disponível é:

$$e_y(k) = r(k) - y(k) \tag{3.41}$$

Como nos algoritmos que foram analisados no capítulo 2, é sempre necessário calcular a derivada do erro em relação aos parâmetros da RN, $\frac{\partial e}{\partial W}$, neste caso obtem-se:

$$\frac{\partial e(k)}{\partial W} = -\frac{\partial y(k)}{\partial W} \tag{3.42}$$

e

[1]Parte-se do princípio que o sinal $r(k)$ possui um conteúdo espectral rico, implicando por isso variações relevantes no sinal $\hat{u}(k)$.

$$\frac{\partial y(k)}{\partial W} = \frac{\partial y(k)}{\partial u(k)} . \frac{\partial u(k)}{\partial W} \tag{3.43}$$

então, para obter a derivada do erro em relação aos parâmetros da RN $\frac{\partial e_y}{\partial W}$, é necessário obter a derivada da saída do sistema em relação à sua entrada $\frac{\partial y(k)}{\partial u(k)}$. Como se pretende identificar o sistema, esta derivada não é possível de obter directamente. Para resolver este problema é utilizada a seguinte aproximação:

$$\frac{\partial y(k)}{\partial u(k)} \approx \frac{\partial \hat{y}(k)}{\partial u(k)} \tag{3.44}$$

o que significa usar o modelo directo do sistema para calcular a derivada da saída do sistema em relação à entrada.

Em relação à figura 3.11 os erros $e_y(k)$ e $e_u(k)$ permitem calcular as seguintes derivadas:

$$e_y(k) \Rightarrow \frac{\partial y(k)}{\partial W} \tag{3.45}$$

e

$$e_u(k) \Rightarrow \frac{\partial u(k)}{\partial W} \tag{3.46}$$

Em resumo, para fazer a propagação deste erro através do Sistema a Controlar seria necessário poder calcular a matriz Hessiana ou a matriz Jacobiana do sistema real, de acordo com o algoritmo de treino escolhido para treinar a rede neuronal. Raramente existe disponível informação suficiente para efectuar estes cálculos o que obriga a recorrer a um modelo directo do sistema a controlar que é colocado em paralelo com o sistema, por forma a ser possível retro-propagar o erro à saída do sistema que se pretende modelizar inversamente, permitindo obter um erro à saída do modelo inverso. O modelo directo terá que ser obtido antes de iniciar o treino do modelo inverso.

O conceito de treino de um modelo inverso especializado tem sido apresentado com algumas variantes e pode, efectivamente, ter algumas diferenças na sua implementação. A figura 3.11 está de acordo com o apresentado em [15]. A figura 3.12 é apresentada em [1] e difere do anterior apenas no facto de ser mais genérico por incluir um modelo de referência. Neste caso o modelo neuronal inverso será um modelo que já inclui um comportamento determinado pelo modelo de referência.

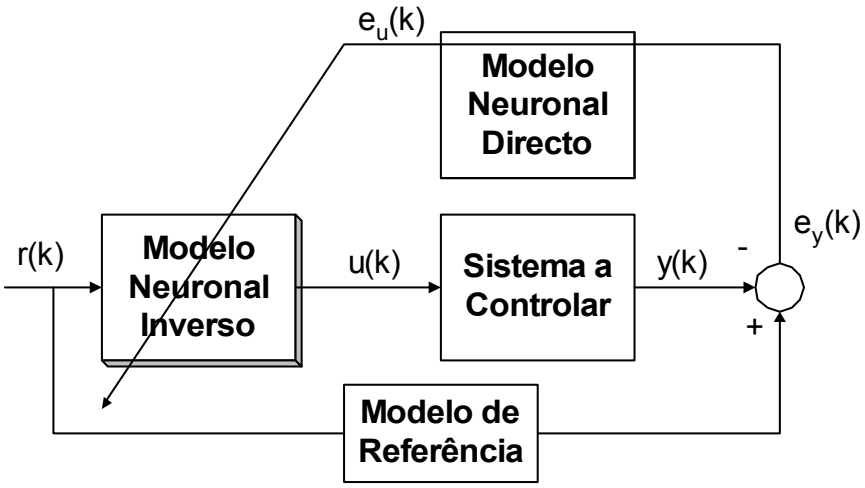


Figura 3.12: Representação do processo de treino de um modelo inverso especializado com inclusão de um modelo de referência.

Outra representação é apresentada em [7] e reproduzida na figura 3.13.

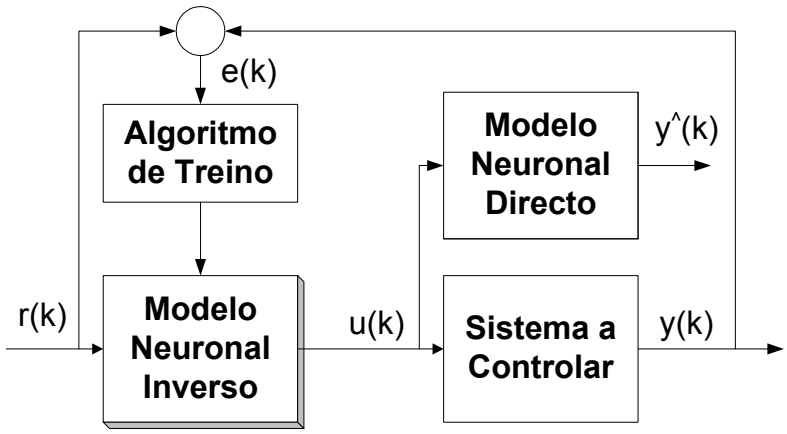


Figura 3.13: Outra representação possível para o treino de um modelo inverso especializado.

Esta representação apesar de ser bastante genérica chama à atenção para a existência de mais do que uma forma de efectivamente implementar o princípio de treino de um modelo inverso especializado. O erro $e_y(k)$ pode ser calculado como a diferença entre a saída do sistema real e a referência ou ser calculado como a diferença entre a saída do modelo directo e a referência. Neste último caso o sistema real seria dispensado o que tornaria o processo bastante mais simples.

No entanto deve notar-se que foi demostrado que, se a saída do sistema real for usada, mesmo que o modelo directo não seja exacto, é possível obter um modelo inverso exacto [54].

Comparando as duas possibilidades para gerar os modelos inversos pode dizer-se que os modelos inversos especializados têm as seguintes vantagens [7]:

- O procedimento é dirigido ao objectivo.
- Mesmo para sistemas que não sejam bi-unívocos é obtido um modelo inverso correcto.

Apesar das vantagens apontadas, os modelos especializados comportam também algumas desvantagens, nomeadamente:

- Apesar de ter sido demonstrado que funciona correctamente, mesmo quando o modelo directo não é exacto, este método é difícil de utilizar com sistemas afectados por ruído.
- A estrutura necessita, em princípio, de um algoritmo de treino iterativo. Isto limita a sua utilização a algoritmos que disponham de tal versão.

Indirect Inverse Adaptation

A figura 3.14 ilustra, de uma forma muito simplificada, outra técnica para gerar modelos inversos, designada por *Indirect Inverse Adaptation* (IIA) [6]. A representação desta técnica não separa claramente as fases de treino e de controlo sendo apenas o controlador usado na fase de controlo e o modelo inverso usado na fase de treino.

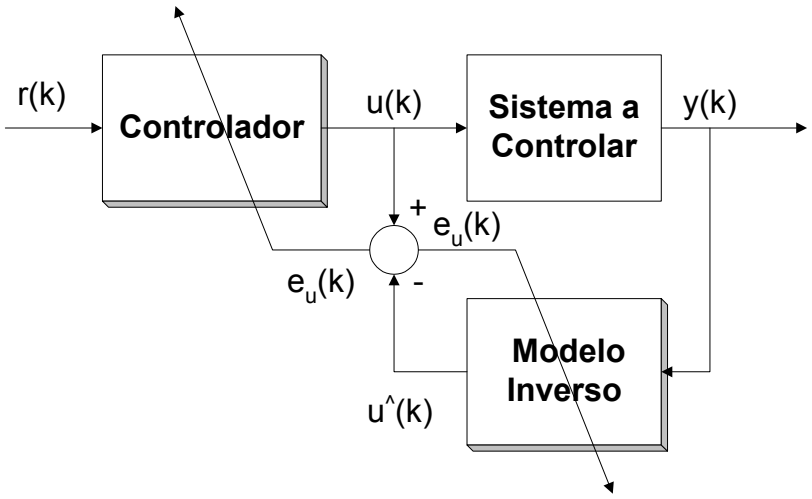


Figura 3.14: Diagrama de blocos do *Indirect Inverse Adaptation.*

Nesta estrutura um modelo inverso (que pode ser neuronal ou não) é usado em vez de um modelo directo como forma de retro-propagar o sinal de saída. O modelo inverso permite estimar $\hat{u}(k)$ a partir da saída do sistema a controlar. Isto permite o cálculo do

erro $e_u(k)$, que será usado para treinar o controlador e o modelo inverso. Como pode ser visto na figura 3.15, que representa o *Controller Output Error Method* (COEM) proposto por [6], a estrutura do *Indirect Inverse Adaptation* tem uma estreita relação com o COEM. De facto, o COEM é um caso particular do IIA, no qual o modelo inverso e o controlador são o mesmo elemento.

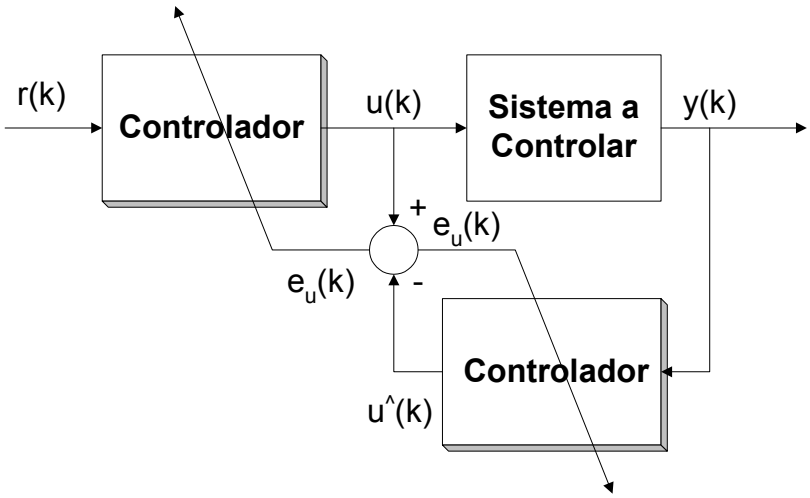


Figura 3.15: Diagrama de blocos do *Controller Output Error Method*.

Esta estrutura apresenta as mesmas vantagens e desvantagens do treino de modelos inversos especializados.

Técnica híbrida genérica/especializada para modelos inversos

A técnica híbrida genérica/especializada para modelos inversos, uma contribuição original desta tese (apresentada em [55], [56] e [57]) , envolve os processos de treino e validação que são usados de forma combinada e é constituída pelos seguintes passos:

1.Criação de um modelo directo.

2.Treino do modelo inverso usando uma estrutura de Modelo Inverso Genérico.

3.Avaliação da qualidade do modelo inverso utilizando uma simulação de controlo com modelo inverso (ver capítulo 5), sendo o sistema substituído pelo modelo directo.

4.Utilização do valor de erro da simulação de controlo como índice de qualidade.

Esta solução, apesar de muito simples e intuitiva, pode ser usada tanto para processos de treino *on-line* como *off-line* e, como será mostrado, melhora a qualidade dos modelos inversos, sem conduzir a um aumento considerável de complexidade do processo.

Analisando os problemas associados às soluções anteriores é possível verificar que eles são suprimidos:

Estrutura	Vantagens	Desvantagens	Complexidade
Inverso Genérico	→Simples de implementar	→Não é *goal directed*	→Baixa →Alg. *batch*
Inverso Especializado e IIA	→*Goal directed* →Permite obter um modelo inverso mesmo quando o directo não é exacto	→Difícil de usar com sistemas afectados por ruído	→Elevada →Alg. iterativo
Técnica Híbrida	→*Goal directed* →Simples de implementar	→Complexidade	→Elevada (necessita avaliação de vários modelos) →Alg. *batch*

Tabela 3.1: Comparação das estruturas de treino existentes.

- A solução híbrida é dirigida ao objectivo uma vez que os modelos são agora avaliados numa simulação de controlo, o que corresponde exactamente à situação em que os modelos inversos serão utilizados.

- Não existe risco de obtenção de um modelo inverso incorrecto, dado que os modelos serão avaliados numa simulação de controlo.

- Esta solução não necessita de versões iterativas dos algoritmos e portanto não impõe qualquer limitação sobre o algoritmo a ser utilizado.

Embora esta solução possa ser utilizada de forma geral, foi desenvolvida no âmbito da implementação de uma combinação de Algoritmos Genéticos e RNs para a escolha da melhor arquitectura. Esta combinação será descrita no capítulo 4.

Comparação das soluções existentes

As soluções disponíveis para as estruturas de treino são comparadas de forma qualitativa e necessariamente resumida através da tabela 3.1.

Como é possível verificar, o problema base apontado à estrutura genérica relativo ao facto de não ser dirigida ao objectivo é resolvido com as estruturas de modelo Inverso Especializado e IIA, que por sua vez limitam os algoritmos que podem ser utilizados. A Técnica Híbrida permite retirar as restrições impostas mas tem como desvantagem a elevada complexidade, uma vez que é necessário treinar vários modelos para procurar aquele que permite obter a melhor qualidade de controlo. A Técnica Híbrida é bastante útil no caso de procedimentos automatizados de optimização de modelos, uma vez que neste caso a procura das diversas soluções está facilitada.

3.7 Capacidade de generalizar

Após a escolha da estrutura de treino a utilizar e da aplicação do correspondente algoritmo de treino está criado um modelo (directo ou inverso) de um determinado sistema. É necessário agora saber se esse modelo tem qualidade suficiente para a tarefa que se pretende que venha a desempenhar.

Um bom modelo tem que ter capacidade de funcionar em situações diferentes daquelas em que foi construído, ou seja, ser dotado de capacidades de generalização (do inglês *generalization*). No caso específico das RNs esta capacidade pode ser verificada aplicando à entrada da RN um sinal diferente daquele que foi usado durante a fase de treino. Se o erro obtido é da mesma ordem de grandeza que o erro de treino, a RN tem uma boa capacidade de generalizar, caso contrário, é possível concluir que a RN aprendeu características específicas do sinal usado na fase de treino ou do ruído que afecta o sistema.

Esta situação (habitualmente designada por treino excessivo - do inglês *overtraining*) é especialmente frequente quando o sinal de treino vem acompanhado de ruído. Na fase inicial a RN aprende as características do sistema, aprendendo depois características do ruído presente no sinal.

3.7.1 Qualidade do modelo

A análise de qualidade do modelo está presente em diversos trabalhos como [38] e [1] e será apresentada aqui de forma próxima à análise feita em [51].

Considerando um modelo genérico implementado por uma RN (o estudo original é mais abrangente, embora o trabalho presente diga respeito apenas a RNs):

$$y(k) \simeq g(\varphi(k), \theta) \tag{3.47}$$

ou seja um modelo com parâmetros ajustáveis θ e que utiliza os regressores $\varphi(k)$ (que podem ser apenas entradas e saídas anteriores). Partindo do princípio que está disponível um conjunto finito de pares entrada-saída de dimensão N, temos:

$$Z_e^N = \{(y(k), \varphi(k)) : k = 1, ..., N\} \tag{3.48}$$

A sequência Z_e^N é designada por sequência de treino ou de estimação, uma vez que o modelo estimado será criado com base nesta sequência.

Desta forma, estimar ou treinar um modelo significa minimizar o erro entre a saída do sistema e a do modelo:

$$\min_{\theta} V_N(\theta, Z_e^N) = \frac{1}{N}. \sum_{k=1}^{N} \|y(k) - g(\varphi(k), \theta)\|^2 \tag{3.49}$$

ou seja, escolher os valores dos parâmetros θ que minimizam o valor do erro.

Supondo que os dados contidos na sequência Z_e^N estão afectados por ruído é possível descrever a saída como:

$$y(k) = g_0(\varphi(k)) + e(k) \tag{3.50}$$

sendo g_0 o modelo "real" e desconhecido do sistema e $e(k)$ o ruído que afecta o sistema.

Medidas da qualidade do modelo

Existem várias formas de medir a qualidade dos modelos desenvolvidos para diferentes aplicações. No presente documento será utilizada a seguinte medida:

$$\bar{V}(\theta) = E\left\|g_0(\varphi(k)) - g(\varphi(k),\theta)\right\|^2 \tag{3.51}$$

onde E(x) é o valor esperado da variável x e $\varphi(k)$ é assumido como estando em regime estacionário.

A medida anterior pode também ser aproximada como a média das amostras:

$$\bar{V}(\theta) = \lim_{N\longrightarrow\infty} \frac{1}{N}.\sum_{k=1}^{N} \left\|g_0(\varphi(k)) - g(\varphi(k),\theta)\right\|^2 \tag{3.52}$$

Naturalmente esta medida depende dos regressores utilizados.

Dentro da estrutura de modelo escolhida, é possível definir o melhor modelo de acordo com a medida de qualidade definida:

$$\theta_*(dm) = \arg\min_{\theta} \bar{V}(\theta) \tag{3.53}$$

onde a dependência explícita dos parâmetros com dm, a dimensão do modelo, é enfatizada.

A medida da qualidade de um dado modelo, denotado pelo sinal *, será obtida por:

$$E\bar{V}(\hat{\theta}_N) = V_*(dm) \tag{3.54}$$

Esta medida reflecte a aptidão esperada do modelo em relação ao sistema, quando aplicada uma nova sequência, sendo $\hat{\theta}_N$ os parâmetros estimados sobre uma sequência de dimensão N.

Usando as definições anteriores é possível utilizar um resultado apresentado em [38] e retomado em [51]. Partindo do princípio que $\hat{\theta}_N$ é resultado da minimização de uma expressão do tipo da proposta na equação 3.49 e que existe um modelo de elevada qualidade denotado por $\theta_*(dm)$ então é possível definir:

$$V_*(dm) \simeq E\left\|g_0(\varphi(k)) - g(\varphi(k),\theta_*(dm))\right\|^2 + E\left\|g(\varphi(k),\theta_*(dm)) - g(\varphi(k),\hat{\theta}_N)\right\|^2 \tag{3.55}$$

O primeiro termo da equação anterior é identificado como sendo o erro de deslocamento (do inglês *bias*) e o segundo o erro de variância.

O erro de deslocamento é devido à estrutura insuficiente do modelo escolhido. Por exemplo se o modelo escolhido não possuir pesos suficientes para o sistema que pretende modelizar, a convergência será para um conjunto de pesos diferente de $\theta_*(dm)$ [1]. O erro de variância é devido a $\theta_*(dm) \neq \hat{\theta}_N$, ou seja é devido ao modelo obtido não ser o melhor modelo possível dentro da dimensão escolhida e pode explicar-se pelo facto de a sequência de treino conter ruído e ser limitada em tamanho.

A existência de erro de deslocamento é inevitável, sendo evidente pela sua natureza que quanto maior for a RN menor será este tipo de erro [1]. Este resultado levaria por si só a aumentar o tamanho da RN, mas o erro de variância aumenta justamente com o número de parâmetros. Isto corresponde ao dilema deslocamento-variância (do inglês *bias-variance dilema*) frequentemente referido na literatura [1], [51] e ilustrado na figura 3.16.

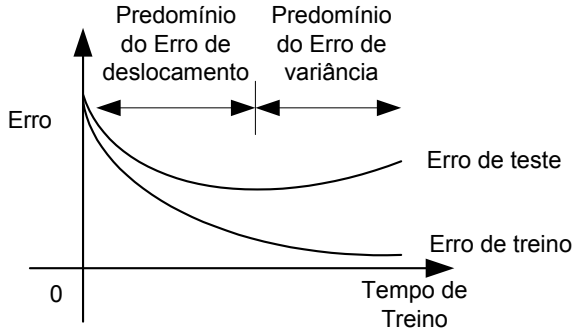


Figura 3.16: Ilustração da situação que origina o dilema deslocamento-variância.

Analisando o comportamento de um modelo durante a fase de treino é possível verificar que, como seria previsível, o critério $\bar{V}(\theta)$ tem um comportamento monótono decrescente sobre a sequência de treino. Sobre a sequência de teste o comportamento é diferente. Na fase inicial o critério baixa e a partir de um determinado número de iterações aumenta. Na realidade a primeira fase corresponde a uma maior importância do valor do erro de deslocamento e a segunda, após a diminuição do erro de deslocamento, à maior influência do erro de variância. Como foi referido no início desta secção este comportamento pode ser explicado como correspondendo à aprendizagem das características do sistema (1ªfase) e à aprendizagem das características do erro ou da sequência de treino (2ªfase).

A aprendizagem de características específicas de um sinal é normalmente indesejável, uma vez que o modelo deixa de ter validade em situações diferentes daquela em que foi treinado, pelo que existem técnicas para evitar este comportamento.

3.7.2 Regularização

Na situação descrita no ponto anterior é possível verificar que na primeira fase são usados alguns parâmetros para fazer a aprendizagem das características do sistema e, na

segunda fase, parâmetros que não são essenciais "aprendem" características específicas do sinal de treino e do erro. Isto significa que existem parâmetros "supérfluos" e embora possa parecer que esta situação é fácil de evitar diminuindo o número de neurónios, na realidade não é [51]. Como evitar então que um modelo perca qualidade por excesso de treino?

Uma das forma de evitar este problema é designada por regularização e consiste em modificar o critério a minimizar adicionando-lhe um termo de penalização:

$$\Omega_N(\theta, Z_e^N) = V_N(\theta, Z_e^N) + \delta \left\|\theta\right\|^2 \tag{3.56}$$

onde δ, que é designado por parâmetro de regularização ou *weigh decay*, é um número pequeno. Em que medida é que esta alteração influencia o comportamento da RN?

Analisando o novo critério é possível verificar que o termo adicionado tem pouca influência se os pesos tiverem valores muito pequenos, introduzindo uma espécie de desvio para zero. Além disso, o valor de δ ser baixo implica que a primeira parte do critério seja mais importante e a influência da segunda só se faça sentir quando o essencial do comportamento do sistema já foi apreendido.

Partindo do princípio que se está a treinar um modelo cuja dimensão é superior à necessária, o que corresponde a uma situação comum, uma vez que a dimensão correcta é desconhecida, numa fase inicial são actualizados os parâmetros que maior influência têm no desempenho da RN. A alteração do critério praticamente não introduzirá nenhuma alteração nesta fase. Na segunda fase os pesos cuja importância é menor deverão ser mantidos próximo de zero, pela influência do segundo termo, devido à baixa influência que a sua actualização teria na função de custo a minimizar.

Como apenas as principais características são aprendidas, o treino excessivo é evitado.

3.7.3 Paragem de Treino Antecipada

Uma alternativa ao método de Regularização é a técnica de Paragem de Treino Antecipada (do inglês *Early Stopping*) que consiste, como o próprio nome indica, em limitar o número de iterações usadas no treino da RN. Supondo que uma procura iterativa é iniciada em $\hat{\theta}_0$, as primeiras iterações irão modificar inicialmente os parâmetros que maior influência têm no erro à saída da RN, como é possível verificar através da análise do funcionamento dos algoritmos de treino baseados em derivadas. Só posteriormente os parâmetros com menor influência, ou os ajustes mais finos, serão efectuados. Terminando o processo de optimização "entre" estas duas fases é equivalente ao processo de regularização uma vez que as características essenciais são captadas, sendo evitada a actualização de parâmetros de menor influência.

A este processo é também chamado Regularização Implícita por oposição ao processo anterior que seria designado por Regularização Explícita [51] e um dos primeiros trabalhos que referiu esta solução, ainda sem utilizar o nome de *Early Stopping* foi publicado por Morgan e Boulard em 1990 [58].

Em [59] Sjöberg demonstrou que a Paragem de Treino Antecipada e a Regularização são formalmente equivalentes. O estudo prático desta questão será abordado com um exemplo, porque embora formalmente equivalentes em cada uma das técnicas existe um parâmetro a estimar. No caso da Paragem de Treino Antecipada é preciso obter o número de iterações que devem ser usadas para treinar a RN, enquanto no caso da Regularização é necessário escolher de forma correcta o valor do parâmetro δ. Não existindo uma forma determinística de calcular qualquer um dos valores, torna-se obrigatório fazê-lo por tentativa e erro.

3.7.4 Outras técnicas

Existem outras técnicas destinadas a resolver o mesmo problema, nomeadamente técnicas de poda (do inglês *pruning*). As mais conhecidas são *Optimal Brain Damage* (OBD) [60] and *Optimal Brain Surgeon* (OBS) [61]. Ambas as técnicas têm uma parte idêntica, daí a designação comum de técnicas de poda, que diz respeito ao princípio de funcionamento: ambas propõem o corte de neurónios como forma de reduzir a complexidade das RNs e evitar o *overtraining*.

A diferenciação das técnicas traduz-se ao nível dos pesos a escolher para serem podados. Naturalmente o objectivo é escolher os pesos que sejam menos úteis ao conjunto formado pela RN, mas as medidas usadas para fazer a avaliação do impacto que a eliminação de determinado peso tem no desempenho da RN são diferentes. Ambas as técnicas foram inicialmente desenvolvidas para usar como base de avaliação o erro de treino, tendo sido proposto em [62] a extensão para usar um critério de avaliação regularizado e em [63] a utilização do erro de generalização em vez do erro de treino.

Estas técnicas obrigam a retreinar as RNs [1] após a poda (embora o OBS sugira a forma de corrigir os pesos dos neurónios que ficam na RN) e são computacionalmente exigentes. Atendendo a estas duas desvantagens estas técnicas são habitualmente menos usadas do que a Regularização e a Paragem de Treino Antecipada.

Outras variantes das técnicas de *prunning* podem ser encontradas nos artigos [64] e [65].

Em [64] é proposta uma solução designada Esqueletização (do inglês *Skeletonization*) que propõe retirar os neurónios que têm menos efeito no erro de saída.

Em [65] a poda é baseada na Análise dos Componentes Principais de acordo com [66], sendo interessante verificar que esta solução de poda pode ser utilizada em conjunto com os métodos de regularização.

No artigo [67] é apresentado um apanhado das técnicas de poda (aqui tomando o termo no sentido genérico), uma vez que inclui a regularização e outras técnicas utilizadas para manter a complexidade da RN limitada de forma a obter uma boa capacidade de generalização - feito numa época relativamente inicial do desenvolvimento destas técnicas.

3.8 Estado da arte

A identificação de sistemas com RNs continua a ser objecto de estudo da comunidade científica, sendo publicados numerosos artigos sobre esta área.

Em [68] o parâmetro μ do algoritmo de Levenberg-Marquardt é analisado como um termo de regularização e são propostas alterações à forma da sua determinação de modo a minorar as possibilidades de *overtraining*.

O artigo [69] estuda o problema da generalização em RNs com elevado número de neurónios, treinadas com o algoritmo do *Steepest descent*. Os autores afirmam que o fenómeno de *overfitting* não é habitualmente global pelo que é sugerido que a paragem de treino antecipada é mais vantajosa do que a regularização.

Sobre a questão da Regularização, embora não se tratando de trabalhos recentes, vale a pena ler os artigos [70], onde é explicado o efeito da regularização e [71], em que é utilizada uma variante da regularização que usa um termo de comparação para o tamanho dos pesos.

Em [72] é apresentado um algoritmo que propõe uma forma de actualizar, iterativamente, o parâmetro de regularização. O algoritmo proposto, designado por *A second derivative of validation error based regularization algorithm* (SDVR), consiste na actualização do parâmetro de regularização de forma semelhante ao que é feito com o algoritmo de Levenberg-Marquardt para os pesos, usando neste caso o erro obtido sobre a sequência de teste. O algoritmo é proposto com duas variantes *convergent updating* e *conditional updating* que diferem principalmente no momento em que procedem à alteração do parâmetro de regularização.

A questão do número ideal de neurónios a utilizar para cada modelo merece também com frequência a atenção da comunidade científica, sendo [73] apenas um exemplo. Neste artigo a estratégia apresentada para determinar o número ideal de neurónios é composta por duas fases: a primeira consiste em escolher um conjunto de modelos, com base numa avaliação do *well-conditioning* do Jacobiano e a segunda consiste na utilização de testes de *Fisher* para avaliar o modelo com menor número de parâmetros, que têm todos significado. Ou seja, a determinação do número ideal de neurónios é feita com base em estimar um conjunto alargado de modelos e na determinação do melhor desses modelos.

A automatização da criação de modelos utilizando validação cruzada é abordada em [49]. Neste trabalho são estudados critérios para avaliar a situação de *overtraining* por forma a incorporar a técnica de *Early Stopping* em procedimentos de treino automatizados. A necessidade de utilizar estes procedimentos automatizados surge da constatação de que, ao contrário do que frequentemente aparece referido noutros trabalhos, as curvas de erro de generalização (sobre uma sequência de teste, ou de validação) possuem vários mínimos. Neste trabalho é também referido que apesar da existência de vários mínimos o melhoramento introduzido pelo estudo alargado da curva de erro de generalização é normalmente baixo, pelo que deverá ser analisada a situação em causa para verificar se o acréscimo de tempo necessário para esse estudo é compensador.

A selecção de modelos criados com RNs, com base em estratégias que sejam suportadas por conceitos estatísticos, é abordada no artigo [74].

3.9 Conclusão

Neste capítulo são apresentadas as fases necessárias à criação de modelos com base em RNs do tipo FNN.

Essas fases passam pela aquisição de dados, preparação dos dados, escolha de uma classe de modelos a utilizar, determinação da ordem do sistema, escolha da estrutura de treino, treino e avaliação da qualidade do modelo.

Na secção referente à escolha de uma classe de modelos a utilizar foram apresentadas algumas das classes usadas no controlo linear e a sua adaptação às RNs e apresentada a classe NNARX como a classe mais apropriada para a modelização com RNs.

Na secção referente à capacidade de generalizar foram abordados problemas específicos dos modelos neuronais, em concreto o *overtraining* e analisadas as formas de tornear esse problema.

Apesar de se pretender dar uma perspectiva o mais completa possível da forma de fazer uma identificação de um sistema, existem questões que ficam em aberto para serem resolvidas com a experiência do utilizador ou para serem objecto de outros estudos. Não existem formas determinísticas de resolver as questões relativas ao número de neurónios a utilizar num modelo, ao número de iterações a usar no treino ou ao valor do parâmetro de regularização a aplicar.

Capítulo 4

Redes Neuronais e Algoritmos Genéticos

"Every new body of discovery is mathematical in form, because there is no other guidance we can have." - Charles Robert Darwin, escritor e naturalista. (1809 - 1882)

4.1 Introdução

Os Algoritmos Genéticos (AGs) surgem no presente trabalho apenas como uma técnica de optimização que pode ser útil para resolver as dificuldades de cálculo de determinados parâmetros. No capítulo 3 verificou-se que existem vários parâmetros necessários para criar um modelo, para os quais não existe uma forma de cálculo analítica, como são os casos do número de neurónios, do número de iterações de treino, do parâmetro de regularização ou até do número de regressores.

Os AGs e as RNs são técnicas computacionais inspiradas nos sistemas biológicos. Uma grande parte da arquitectura neurológica dos sistemas biológicos é determinada geneticamente, pelo que não é surpreendente que à medida que os investigadores da área das RNs exploraram a forma como os sistemas neuronais biológicos estão organizados surgisse a ideia de criar RNs evolutivas [75]. É principalmente neste âmbito que a técnica de optimização designada por AG surge neste trabalho, não sendo no entanto um objecto central de estudo, pelo que será introduzida de forma sucinta e tendo em vista o essencial para o desenvolvimento do restante trabalho.

Os AGs são métodos de procura adaptativos utilizados na pesquisa de soluções e na resolução de problemas. A sua inspiração está nos processos biológicos dos organismos vivos, nomeadamente no princípio da sobrevivência do mais forte (ou do mais apto), de acordo com o princípio da evolução de Charles Darwin [76].

A ideia subjacente aos AGs é reproduzir o processo evolutivo da natureza. Num processo iterativo, através de diversas gerações, o algoritmo faz evoluir uma população que é constituída por soluções candidatas para um dado problema. A população inicial consiste, em geral, num conjunto de soluções potenciais distribuídas aleatoriamente no espaço de procura. Através de operadores como a selecção, recombinação e mutação,

os indivíduos vão sendo alterados e evoluindo para zonas mais promissoras do espaço de procura até que uma solução considerada aceitável seja obtida [76].

A avaliação da qualidade dos indivíduos é feita por intermédio de uma função de aptidão ou função objectivo que pretende medir até que ponto cada indivíduo da população serve os objectivos que estão subjacentes à sua evolução.

Aos AG são genericamente apontadas as seguintes características [17]:

- Técnica de optimização que não utiliza derivadas.
- Conceito intuitivo motivado pela inspiração biológica.
- Técnica lenta. Devido ao seu carácter aleatório, as implementações de AG têm muitas vezes que testar um elevado número de soluções, o que as torna lentas. No início esta característica impossibilitou a utilização dos AG na resolução de alguns problemas mas actualmente, com o aumento da capacidade de processamento dos computadores, esta limitação é cada vez menos importante.
- Técnica flexível por permitir a utilização de uma função objectivo tão complexa quanto for necessário.
- Aleatoriedade. Existe nos AG um forte peso da aleatoriedade uma vez que a população é gerada de forma aleatória e alguns dos operadores utilizados para gerar as novas populações têm também na sua forma de implementação algum carácter aleatório.
- Opacidade analítica. Esta técnica é difícil de analisar analiticamente, em parte devido à aleatoriedade, pelo que a maioria do conhecimento foi obtido de forma empírica.
- Natureza iterativa. Ao contrário, por exemplo, dos estimadores de mínimos quadráticos esta técnica é iterativa por natureza necessitando por isso de um critério de paragem. Os critérios de paragem mais comuns têm por base o tempo de computação, um valor predefinido da função objectivo, uma melhoria mínima por iteração (ou conjunto de iterações) ou uma melhoria mínima relativa por iteração (ou conjunto de iterações).

Os AGs como técnica de optimização podem ser usados para obter os pesos de uma RN (como foi referido no capítulo 2, secção da optimização sem derivadas), no entanto dada a elevada precisão necessária, no presente trabalho tal não foi possível de concretizar.

Os AGs foram utilizados no presente trabalho para a optimização da arquitectura de uma RN e para fazer uma comparação prática entre as técnicas de Regularização e Paragem de Treino Antecipada, como será documentado no capítulo 6.

4.2 Algoritmos Genéticos

Algoritmos Genéticos ou algoritmos de procura genéticos são uma técnica de optimização de funções baseada nos princípios de evolução da genética e no processo de selecção natural existente na natureza [77] de acordo com o trabalho pioneiro de John Holland [78].

O objectivo inicial do trabalho de Holland era estudar o fenómeno da adaptação existente na natureza, mas o seu trabalho foi posteriormente utilizado como uma técnica de optimização usando um critério para avaliar a solução mais apta, correspondendo ao princípio da sobrevivência do mais forte.

A criação de uma nova geração é feita através de operadores que modificam os genes da população e operadores que simulam a procriação, dando origem à reprodução. Uma nova geração corresponde à substituição de alguns, ou todos, os indivíduos de uma população por indivíduos criados por reprodução. Na forma mais simples os elementos da população são codificados como sequências de bits, devendo o seu tamanho ser determinado em função da resolução necessária para o problema em análise [76].

Um elemento da população poderá, numa aplicação concreta, ser composto por um conjunto de parâmetros, sendo cada parâmetro um gene do ser em questão.

Se a aplicação fosse uma RN, os parâmetros poderiam ser os pesos associados às ligações e o critério de avaliação dos indivíduos que compõem a população poderia (entre outras possibilidades) ser representado pelas equações 2.3 ou 2.4, usando o erro entre a função real e a saída da RN. Cada indivíduo representaria, através das sequências binárias, os pesos (e de forma implícita ou explícita também a arquitectura da RN) que constituem a RN e a avaliação da qualidade desse indivíduo, para o fim em questão, poderia ser feita através de uma simulação do modelo representado. Essa avaliação seria quantificada usando, por exemplo, as equações 2.3 ou 2.4.

Esta situação está ilustrada na figura 4.1, onde se mostra a codificação de um indivíduo da população e a respectiva interpretação da informação genética nele guardada.

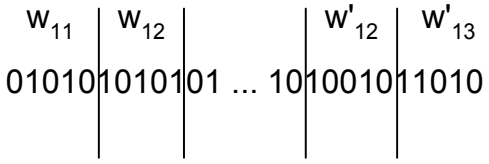


Figura 4.1: Exemplo da codificação de um indivíduo numa optimização com AG e a respectiva interpretação da informação.

Desde o trabalho inicial, muitos novos operadores foram propostos e melhoramentos introduzidos mas o Cruzamento, a Mutação e o Elitismo (respectivamente dos termos em inglês *crossover, mutation* e *elitism*) são os operadores presentes em quase todas as aplicações de AGs para optimização, pelo que se justifica de uma forma simples explicar o seu funcionamento.

4.2.1 Cruzamento

O cruzamento dá origem a novos membros da população através de um processo de mistura de informação genética de ambos os pais. Na figura 4.2 pode ver-se um exemplo de cruzamento num só ponto, na parte superior, e em dois pontos, na parte inferior, podendo ser definidos cruzamentos em N-1 pontos para uma sequência de N pontos.

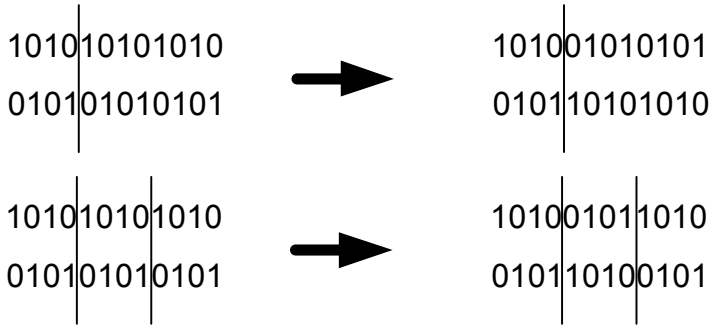


Figura 4.2: Exemplo do operador de cruzamento. Na metade superior num só ponto, na parte inferior em dois pontos.

A operação de cruzamento coloca a questão de qual será a melhor forma de escolher os pais para obter um crescimento mais rápido das aptidões da população. Entre várias outras soluções esta escolha pode ser feita com o método da Roleta, de forma Aleatória e de forma Elitista [78] [79] [76].

O método da Roleta consiste em criar uma roleta representada por um círculo sendo atribuída a cada membro da população uma fatia proporcional à sua aptidão. A roleta é então girada para se obter o conjunto de progenitores necessário, sendo escolhido o progenitor correspondente à fatia do número sorteado pela roleta. Neste caso a probabilidade de determinado progenitor ser escolhido tem a ver directamente com a sua aptidão.

A escolha Aleatória é feita atribuindo um número a cada elemento que constitui a população e sorteando entre esses números escolhe-se o conjunto de progenitores.

A selecção Elitista é feita escolhendo como progenitores os elementos que têm melhores aptidões. Normalmente este tipo de selecção é complementada com outros tipos de selecção para evitar que as soluções encontradas fiquem muito dependentes das soluções iniciais.

4.2.2 Mutação

A mutação é o processo pelo qual uma percentagem dos genes são seleccionados de forma aleatória e modificados. Este operador é uma garantia de que potencialmente todas as soluções podem ser atingidas. Um AG implementado, por exemplo, só com cruzamentos e elitismo está muito dependente da codificação genética inicial dos indivíduos, uma vez que ao elitismo não está associado nenhum mecanismo de alterar

as codificações e o cruzamento apenas permite trocas de porções da sequência binária. Se, na codificação inicial de todos indivíduos o j-ésimo bit de um parâmetro contivesse o mesmo valor, a solução final ficava automaticamente condicionada a manter sempre esse j-ésimo bit com o mesmo valor. Frequentemente a mutação é associada com uma probabilidade e mediante essa probabilidade a modificação genética poderá acontecer ou não. Por exemplo definindo uma probabilidade de mutação de 5% significa que em média num indivíduo codificado com 20 genes, um será modificado por acção da mutação. As implementações mais comuns fazem uso de um gerador aleatório para decidir sobre a modificação de cada bit.

A operação de mutação está ilustrada na figura 4.3.

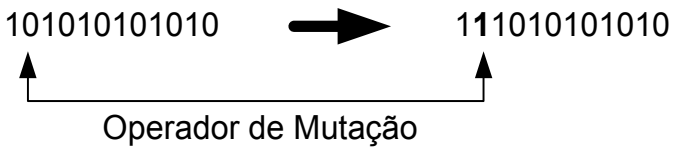


Figura 4.3: Exemplo da operação de mutação.

4.2.3 Elitismo

O elitismo corresponde a guardar para cada nova geração os melhores membros da geração anterior (elite) de forma a garantir que há uma maximização contínua (sem retrocessos) da aptidão da população. A utilização de um processo elitista pode também ter influência na operação de cruzamento, uma vez que as elites podem permanecer na população durante várias gerações. A utilização das elites pode desta forma ter um efeito perverso de condicionar o aparecimento de indivíduos consideravelmente diferentes geneticamente dos que existiam na população inicial.

4.2.4 Função de aptidão

A função de aptidão (do inglês *fitness*) é a função escolhida para classificar os indivíduos. De acordo com estes valores os indivíduos mais aptos poderão ser escolhidos para as operações de reprodução e para serem conservados para a geração seguinte através do elitismo ou serem simplesmente eliminados com a geração seguinte.

A avaliação dos indivíduos envolve normalmente a descodificação das soluções e a avaliação através de processos exteriores ao ambiente de evolução dos AGs. Por exemplo, no caso de se tratar de uma aplicação de AGs que pretenda determinar os pesos de uma RN, para avaliar a aptidão das soluções será necessário descodificar os valores dos pesos de cada indivíduo da população, a partir desses pesos criar e treinar e depois avaliar a qualidade do modelo através de uma simulação, usando finalmente como valor de aptidão o erro entre os valores obtidos pelo modelo e as saídas reais do sistema a modelizar.

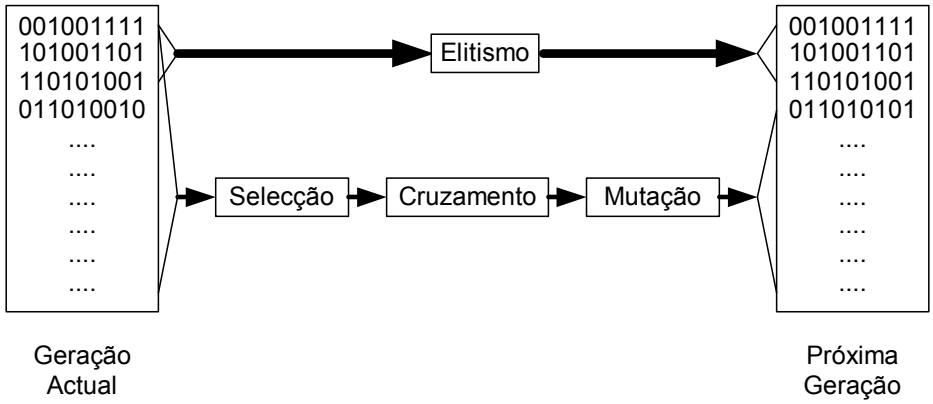


Figura 4.4: Funcionamento do 3^{o}passo da proposta de implementação de AG.

4.2.5 Algoritmo

Um algoritmo possível para a implementação de um AG que inclui Cruzamento, Mutação e Elitismo será:

Passo 1: Escolher a forma adequada de codificar as soluções potenciais, a sua resolução em termos do número de bits, o número de elementos da população e critério adequado para avaliar a população (função de aptidão).

Passo 2: Inicializar todos os elementos da população de forma aleatória e avaliá-los usando o critério escolhido.

Passo 3:

a) Seleccionar dois elementos da população de acordo com o método de selecção escolhido.

b) Aplicar o operador de cruzamento.

c) Aplicar a mutação de acordo com a probabilidade escolhida.

d) Repetir de a) a d) até a nova geração estar completa.

e) Aplicar elitismo repondo os melhores indivíduos da geração anterior.

Passo 4: Repetir os passos 3 e 4 até ser atingido o critério de paragem.

Habitualmente como critério de paragem usa-se o número de iterações ou um valor do critério de avaliação da população.

Para ilustrar o passo 3 do algoritmo podemos utilizar a figura 4.4 [17].

A optimização através dos AG é especialmente útil quando não é possível obter um resultado determinístico para o problema ou a gama de soluções é demasiado ampla para uma procura exaustiva da melhor alternativa e uma solução que contemple um mínimo local possa ser aceitável. Este método tem também a vantagem de ser um método de optimização global e de não correr riscos de ficar "preso" num mínimo local.

4.2.6 Outros operadores

Existem vários operadores, para além dos já referidos, que podem ser usados nas optimizações com AGs, por exemplo sub-populações, operadores de duplicação, operadores de transdução e operadores de conjugação, entre outros.

Sub-populações

Podem ser utilizados conjuntos de sub-populações cujos melhores indivíduos podem integrar as elites ou pode ser-lhes permitido a propagação através das sub-populações para obter optimização global [80].

Operadores de duplicação e deleção

Estes operadores resultam de situações anómalas na natureza em que acontece a duplicação ou supressão de um gene. Nas aplicações mais comuns dos AGs estes operadores não têm grande utilidade, mas são utilizados em casos de programação genética de representação em árvores [76].

Operador de transdução

O operador de transdução é o correspondente computacional de um processo biológico que permite que em determinadas circunstâncias uma determinada característica de um indivíduo possa ser passada à totalidade de uma população. Tal como no caso anterior este operador apenas é útil em casos particulares.

Operador de Conjugação

A conjugação é um mecanismo biológico presente nas bactérias que permite que estas, reproduzindo-se assexuadamente, possuam grande diversidade genética entre os indivíduos da população. O seu funcionamento caracteriza-se pela transferência unidireccional de material genético entre duas células. Este operador, que está ilustrado na figura 4.5, foi incorporado em algumas optimizações com AGs como forma de garantir a diversidade dos indivíduos da população [76].

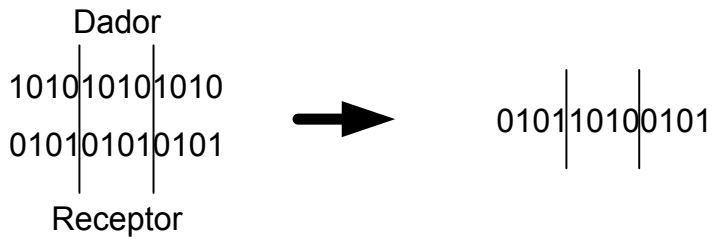


Figura 4.5: Exemplo do operador de conjugação.

Para aplicar o operador de conjugação é necessário definir um indivíduo dador e um indivíduo receptor e os pontos de origem e de fim. A informação genética é então trocada, sendo criado um indivíduo com a informação entre os pontos de origem e de fim do dador e a restante do receptor.

Este operador tem semelhanças óbvias com o operador de cruzamento uma vez que repetindo duas vezes a operação de conjugação e efectuando a troca de dador e receptor são obtidas as duas soluções proporcionadas pelo operador de cruzamento. Por esse facto em algumas aplicações este operador é usado em substituição do operador de cruzamento.

4.3 Redes Neuronais e Algoritmos Genéticos

Como já foi assinalado RNs e AGs têm algumas coisas em comum e é de certa forma natural o aparecimento de várias formas distintas de combinação de ambas as técnicas. A produção tem sido tão profícua que actualmente até a produção de artigos científicos com *surveys* e *reviews* sobre a intersecção destas áreas é já bastante vasta como são exemplos [81], [82], [83] e [75].

Habitualmente os AGs, enquanto técnica de optimização, são utilizados como auxiliar da RN uma vez que nesta existem muitos parâmetros cujos valores devem ser estabelecidos de forma a obter o comportamento pretendido. Estes parâmetros tanto podem ser os próprios pesos da RN como outros parâmetros que definem a arquitectura, nomeadamente o número de entradas, o número de neurónios da camada escondida, o número de iterações de treino, entre outras possibilidades.

Tal como é apontado em [75], são três as principais formas de combinação de AGs e RNs:

- Utilização de AGs para obter os pesos de RNs de arquitectura fixa.
- Utilização dos AGs para escolha da arquitectura da RN.
- Utilização dos AGs para escolha dos dados de treino e para interpretação da saída das RNs.

As duas primeiras possibilidades são as que se revestem de interesse para o presente trabalho, pelo que apenas estas serão analisadas.

A utilização de AGs com RNs de arquitectura fixa, apesar de apenas em casos muito simples permitir a obtenção directa dos valores dos pesos, reveste-se de muitas variantes. Assim, como é reportado em [75], já foram desenvolvidas aplicações para determinar a taxa de aprendizagem do treino e a taxa associada à técnica de momento e também aplicações para usar os AGs apenas como iteração inicial de treino, substituindo os pesos aleatórios por pesos iniciais optimizados com os AGs.

Em [75] é também explicado parte do motivo que leva à complexidade de obter os pesos para uma RN. O motivo apontado é designado por Problema das Permutações (do inglês *Permutations Problem*) e resulta de uma grande parte das RNs serem constituídas por neurónios cuja posição é facilmente permutável. Ou seja se numa RN

trocarmos a ordem dos neurónios ela continua a desempenhar a mesma função, mas a sua descrição em termos de *strings* binárias não será a mesma. Na verdade para uma RN com h neurónios existem até h! soluções idênticas.

A utilização dos AGs para escolha da arquitectura da RN é a forma de combinação de AGs e RNs com maior sucesso tendo conduzido às mais diversas estratégias.

As aplicações mais comuns consistem na codificação da estrutura da RN na forma binária e no treino da rede com base num dos algoritmos discutidos no capítulo 2, sendo usado como função de aptidão o erro obtido pela RN após o treino. Na grande maioria das aplicações o algoritmo usado foi o *steepest descent*, resultando daí um tempo de processamento praticamente proibitivo mesmo para aplicações de pequena dimensão.

A grande maioria das variantes surgem ao nível da codificação que será abordada na secção seguinte, da aplicação de funções a minimizar mais complexas e até do desenvolvimento de aplicações que não só optimizam a estrutura como os pesos.

4.3.1 Estratégias de codificação

O artigo [84] apresenta uma síntese das estratégias de codificação utilizadas neste domínio, divididas pela codificação de pesos e pela codificação das estruturas das RNs.

Codificação dos pesos

A codificação de pesos mais simples diz respeito à utilização de *strings* binárias justapostas para formar uma *string* que representa o conjunto dos pesos da RN. No entanto existem soluções mais elaboradas que são mais próximas da realidade do cérebro humano, como por exemplo a existência de um bit que assinala a efectividade da ligação, ou de dois bits para assinalar que a ligação está desligada, é inibidora ou excitatória [84].

Nas muitas aplicações já feitas nesta área é possível encontrar os pesos codificados como *strings* de bits, números reais e *strings* de bits em codificação de Grey. Existem também aplicações em que o número de bits utilizados na codificação é aumentado durante a evolução por forma a permitir um ajuste fino numa fase mais avançada [84].

Codificação da arquitectura da RN

A tarefa de codificar uma arquitectura de RN para utilização com AGs é consideravelmente mais complexa do que a codificação dos pesos uma vez que permite muitas soluções diferentes e com maior ou menor eficácia. Como será possível concluir da classificação apresentada em [84] existe informação que está representada de forma explícita na codificação e outra que apenas está de forma implícita.

- Codificação baseada nas ligações: neste tipo de codificação (que constitui a maioria das aplicações iniciais) a informação presente no genoma de cada indivíduo é relativa apenas às ligações. Neste tipo de codificação existe normalmente uma arquitectura máxima, que é disposta em camadas ou com todas as ligações presentes.

- Codificação baseada em nós: neste tipo de codificação a base é o nó, podendo existir informação sobre a posição relativa, ligação a nós anteriores, pesos e função de activação. Neste caso as operações de cruzamento e mutação têm que ser efectuadas com cuidados adicionais.

- Codificação baseada em camadas: a codificação é neste caso baseada nas camadas através da descrição das ligações entre elas e resulta numa complexidade acrescida que exige operadores especiais.

- Codificação baseada em caminhos: este tipo de codificação é utilizada para RNs com realimentação e as redes são vistas como um conjunto de caminhos do nó de entrada para o nó de saída. Também neste caso são necessários operadores especiais.

- Codificação indirecta: o princípio base desta codificação é utilizar uma gramática própria para codificar as RNs em vez de codificar directamente as propriedades da RN.

4.4 Conclusão

Este capítulo faz uma curta introdução à técnica computacional designada por Algoritmos Genéticos. Esta técnica resultante de inspiração biológica correspondente à sobrevivência do mais forte (ou do mais apto), tem como grande vantagem ser uma técnica de optimização global (permite, teoricamente, a obtenção de qualquer solução existente no espaço de procura). São explicados os princípios base que a constituem e é exemplificada uma estrutura de algoritmo que usa os operadores mais comuns.

Neste capítulo é também abordada a combinação de AGs e RNs e as formas mais comuns que têm constituído essas combinações. Este é um capítulo necessariamente curto uma vez que os AGs não são um objecto central de estudo desta tese, sendo utilizados essencialmente como instrumento de optimização em diversas situações como será documentado no capítulo 6.

Capítulo 5

Estruturas de controlo

"Most people say that it is the intellect, which makes a great scientist. They are wrong: it is character." - Albert Einstein, cientista. (1879 - 1955)

5.1 Introdução

Após terem sido analisados os algoritmos e as técnicas usadas para obter modelos de boa qualidade, directos e inversos, podem agora procurar-se as malhas de controlo que fazem uso desses modelos.

As malhas de controlo presentes na literatura ([7], [1], [52], [15], [53], [6], [85] e [86]) que fazem uso de RNs são diversas sendo algumas específicas para as RNs e outras adaptadas do controlo clássico.

Este capítulo pretende apenas referir os controladores baseados em modelos usados directamente no presente trabalho e alguns outros com afinidades directas com os que foram utilizados.

Neste capítulo surge também a segunda contribuição desta tese que diz respeito à malha do Controlador Aditivo Baseado em Modelo Interno (do inglês *Additive Internal Model Control* - AIMC) que, como será demonstrado ao longo deste documento, tem interesse não só do ponto de vista do controlo, mas também na sua utilização em estruturas de treino *on-line*.

5.2 Controlador com Modelo Inverso

O Controlo com Modelo Inverso (do inglês *Direct Inverse Control* - DIC) é o tipo de controlo mais simples e consiste em ligar em série com o sistema o seu modelo inverso. Como é evidente, se o modelo inverso for de "boa" qualidade, a saída do sistema será igual à entrada de referência do controlador.

Para um sistema genérico que possa ser descrito de forma discreta pela equação:

$$y(k+1) = g[y(k), ..., y(k-n+1), u(k), ..., u(k-m+1)] \tag{5.1}$$

onde n é o número de regressores da saída y e m o número de regressores da entrada u, o modelo inverso pode ser obtido da seguinte forma:

$$\hat{u}(k) = g^{-1}[y(k+1), ..., y(k-n+1), u(k-1), ..., u(k-m+1)] \tag{5.2}$$

onde ^indica que o parâmetro, ou função em causa, é estimado.

A figura 5.1 mostra o diagrama de blocos de um controlador deste tipo.

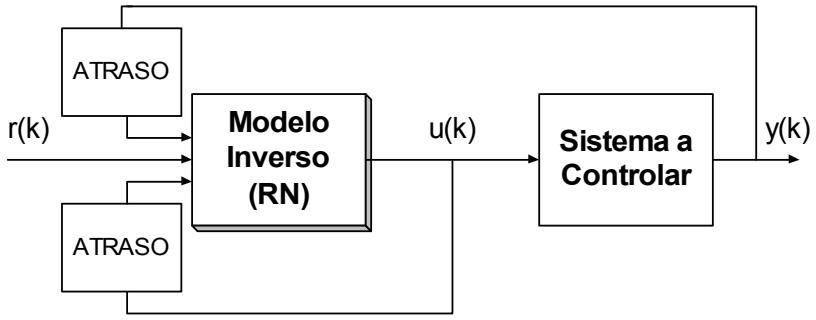


Figura 5.1: Controlador com Modelo Inverso com detalhe sobre as entradas do modelo inverso. $r(k)$ é a referência, ou seja o comportamento que se pretende para o sistema em malha fechada, $u(k)$ o sinal de controlo e $y(k)$ a saída do sistema a controlar.

A obtenção do modelo neuronal inverso, em termos práticos, será feita de acordo com o que foi visto no capítulo 3 no que diz respeito às estruturas de treino e usando os mesmos dados que seriam necessários para o modelo directo.

A principal alteração, no que respeita ao treino, tem a ver com as alterações entre entradas e saídas. É possível verificar o tipo de alterações através de um exemplo: para um sistema em que existem dois regressores para entrada e saída, o diagrama de blocos do modelo directo será o da figura 5.2.

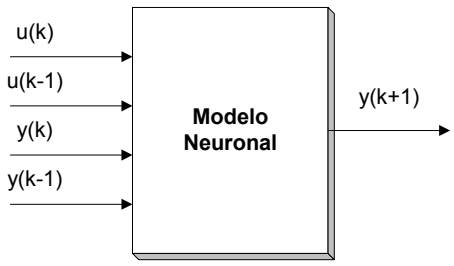


Figura 5.2: Diagrama de blocos do modelo de um sistema de 2ª Ordem.

O correspondente modelo inverso terá a relação de entradas e saídas demonstrada pelo diagrama de blocos da figura 5.3.

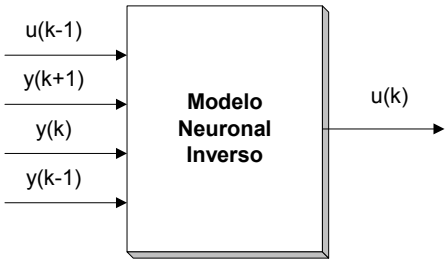


Figura 5.3: Diagrama de blocos do modelo inverso de um sistema de 2ª Ordem.

Nos regressores, $y(k+1)$ por não estar disponível durante a execução da amostra k, é habitualmente substituído pela referência correspondente. Um exemplo desta situação está representado na figura 5.4.

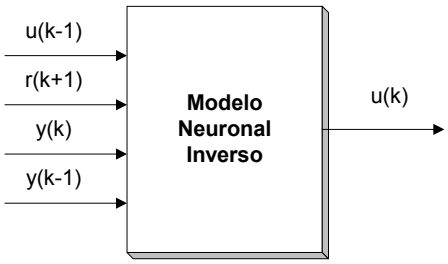


Figura 5.4: Diagrama de blocos do modelo inverso de um sistema de 2ª Ordem, com substituição das amostras de y não disponíveis por amostras da referência.

Neste caso a malha do Controlador com Modelo Inverso correspondente está representada na figura 5.5.

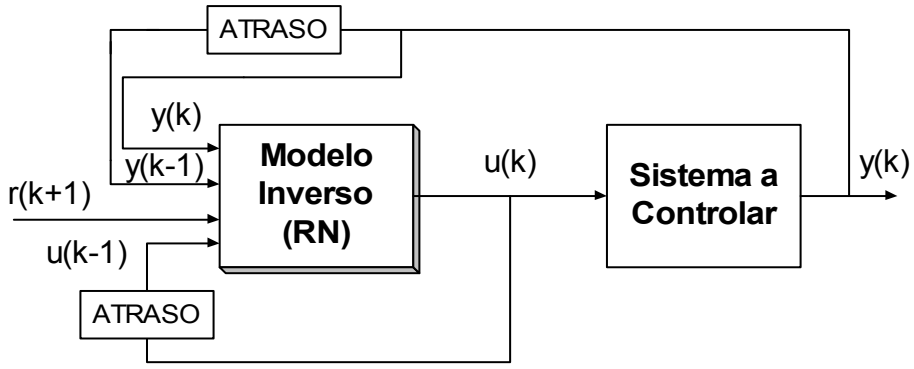


Figura 5.5: Exemplo de um Controlador com modelo inverso, usando um modelo de 2ª ordem.

O controlo com modelo inverso é habitualmente considerado controlo em malha aberta, no entanto tal pode ou não acontecer dependendo da forma como for imple-

mentado. A implementação proposta pela figura 5.5 corresponde a existir realimentação de forma indirecta através das entradas anteriores do sinal de saída $y(k)$. Se fossem utilizadas amostras de $r(k)$ em vez de amostras de $y(k)$ a malha seria completamente aberta. Esta situação não é muito comum, uma vez que se perde informação sobre o valor efectivo da saída.

O controlo com modelo inverso dá normalmente origem a um sinal de controlo muito activo [1], devido à inexistência de realimentação, o que, dependendo do sistema em causa, pode ser inconveniente.

5.3 *Controller Output Error Method e Indirect Inverse Adaptation*

O COEM[6], como foi afirmado no capítulo 3, tem uma grande proximidade com a estrutura de treino e controlo *Indirect Inverse Adaptation.* A principal diferença resulta do facto de o modelo inverso utilizado para obter o sinal $\hat{u}(k)$ ser o mesmo modelo usado como controlador.

As figuras 5.6 e 5.7 representam respectivamente o IIA e o COEM.

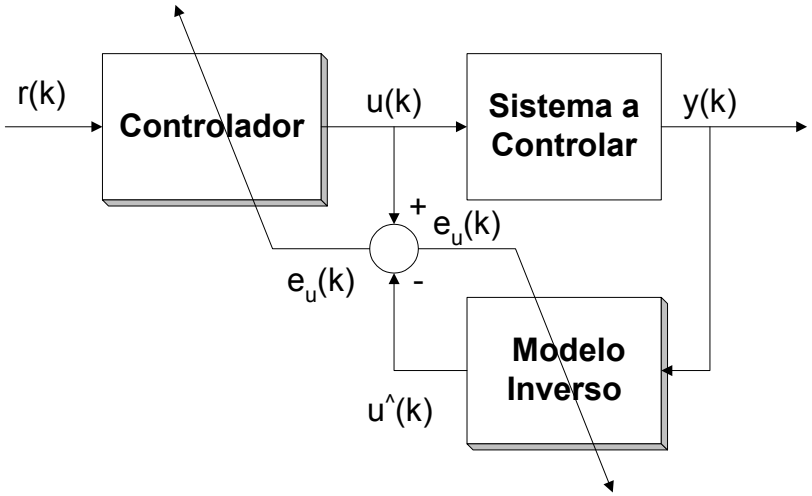


Figura 5.6: Diagrama de blocos do *Indirect Inverse Adaptation.*

Pelo exposto e pela análise das figuras é fácil verificar que o COEM é um caso particular do *Indirect Inverse Adaptation* e que em ambos os casos o controlo efectuado será semelhante ao Controlo com Modelo Inverso analisado na secção anterior.

5.4 Controlador Aditivo

O princípio de funcionamento do Controlador Aditivo (do inglês *Additive Feedforward Control - AFC)* é bastante simples: adicionar a um controlador com realimentação, mas

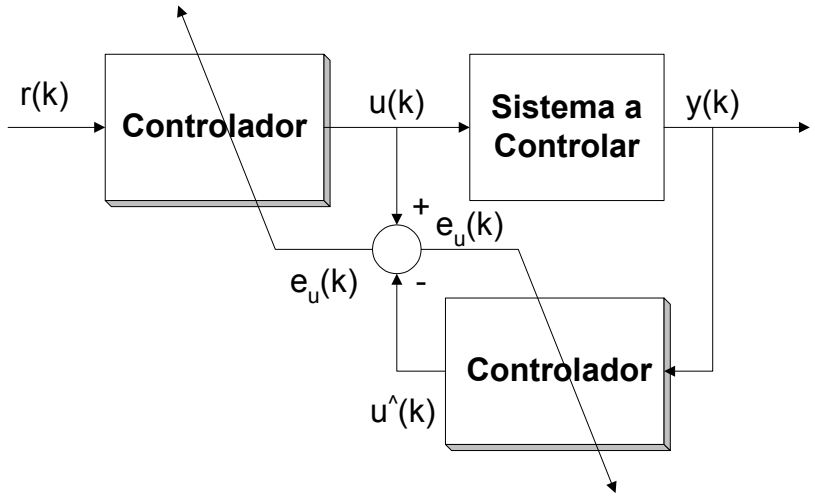


Figura 5.7: Diagrama de blocos representativo do *Controller Output Error Method.*

cujo funcionamento não é considerado satisfatório, um controlador adicional constituído por um modelo inverso do sistema. Este princípio de funcionamento está ilustrado na figura 5.8.

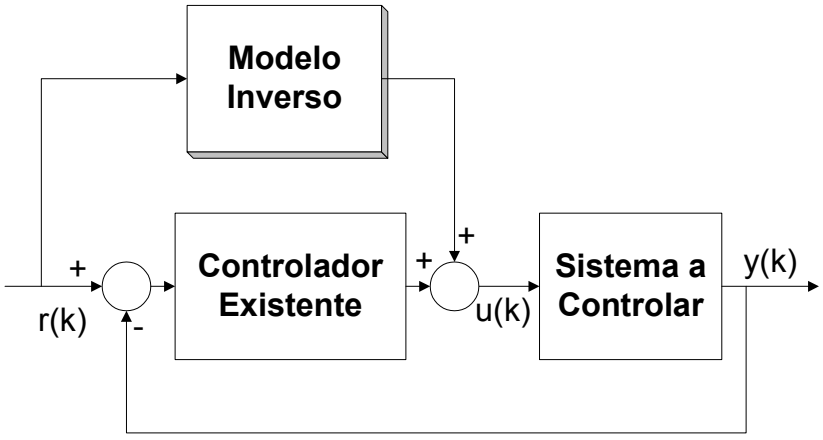


Figura 5.8: Estrutura usada para o Controlo Aditivo. Os sinais $r(k)$, $u(k)$ e $y(k)$ têm o mesmo significado que anteriormente.

Enquanto o controlador que já actuava sobre o sistema recebe como sinal de entrada a diferença entre o sinal de saída e a referência, o novo controlador recebe directamente o sinal de referência. Os sinais de saída de ambos os controladores são adicionados constituindo o sinal de controlo que é aplicado ao sistema. Isto permite que o controlador existente funcione com base no erro e que o modelo inverso assuma o controlo da malha, uma vez que, se este modelo apresentar um bom mapeamento inverso do sistema, o erro será quase nulo e o controlador existente não terá praticamente nenhum efeito

sobre a malha. Daqui resulta também a facilidade de remoção do controlador existente.

A estratégia de Controlo Aditivo apresenta as seguintes vantagens [15]:

- A recolha de informação pode ser feita sem abrir a malha de controlo existente, evitando a paragem de funcionamento do sistema e facilitando o acesso a dados de boa qualidade (para preparação de um modelo inverso), uma vez que serão recolhidos na zona de operação correcta.

- Não existe necessidade de abrir a malha de controlo nem durante a fase de treino nem para fazer a introdução do novo controlador.

Dentro da estrutura do Controlador Aditivo é possível distinguir duas formas de fazer a implementação [15] que correspondem a diferentes utilizações da saída do Sistema a Controlar e da referência.

É designado como Controlador Aditivo Puro a implementação que não utiliza uma realimentação através dos valores anteriores das entradas do modelo inverso e é designado por Controlador Aditivo Misto a implementação que usa realimentação através dos valores anteriores das entradas do modelo inverso. Estas designações são lógicas uma vez que num controlador aditivo não se supõe a existência de realimentação que não seja a da malha pré-existente.

5.4.1 Controlador Aditivo puro

O Controlador Aditivo Puro, que está representado na figura 5.9, usa como informação para as entradas do modelo inverso apenas a referência e a saída do próprio modelo inverso. Assim sendo apenas existe realimentação na malha do controlador existente.

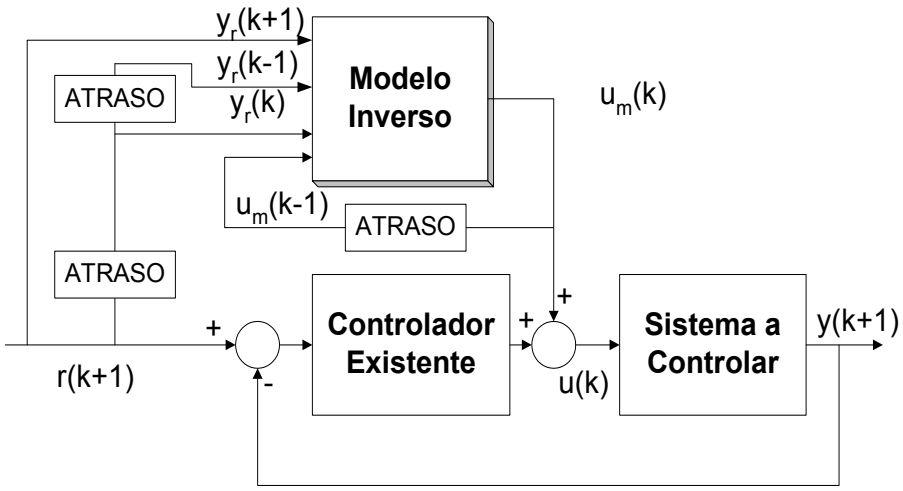


Figura 5.9: Diagrama de blocos de um Controlador Aditivo Puro.

5.4.2 Controlador Aditivo misto

O Controlador Aditivo Misto, representado na figura 5.10 difere do anterior apenas pelo facto de utilizar como entrada do modelo inverso valores da saída do sistema.

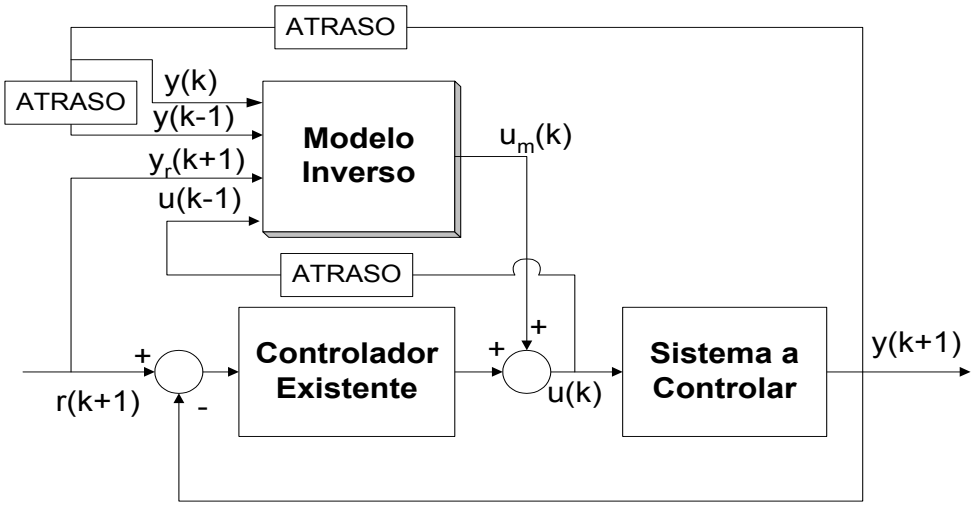


Figura 5.10: Diagrama de blocos de um Controlador Aditivo Misto.

Para ambas as formas de implementar o Controlador Aditivo é possível obter controlo de boa qualidade, não sendo a realimentação nas entradas do modelo inverso estritamente necessária uma vez que já existe através da malha do controlador existente. No entanto, o controlador aditivo puro é mais "exigente" em relação ao modelo inverso do sistema uma vez que, não existindo realimentação, é necessário que este modelo seja "mais perfeito". Deve notar-se que o tipo de controlo efectuado é muito semelhante ao controlo com modelo inverso, excepto pela eventual contribuição da malha com realimentação.

O princípio do controlador aditivo levanta novamente a questão discutida em 3.3.1, uma vez que o controlador existente não estará a funcionar afectado por um factor de escala como acontecerá com o modelo inverso que será adicionado à malha existente (excepção feita a uma eventual situação de teste deste tipo de malha).

5.5 Controlador Baseado em Modelo Interno

O Controlador Baseado em Modelo Interno (do inglês *Internal Model Control* -IMC) [87] [88] é uma estrutura que possibilita que o sinal de realimentação reflicta os efeitos das perturbações que afectem o sistema e as diferenças entre o modelo e o sistema real.

O diagrama de blocos da figura 5.11 representa a malha de um Controlador Baseado em Modelo Interno "clássico".

A partir do diagrama de blocos, considerando o domínio de Laplace, supondo que S descreve o comportamento do sistema a controlar, M^{-1} descreve o comportamento

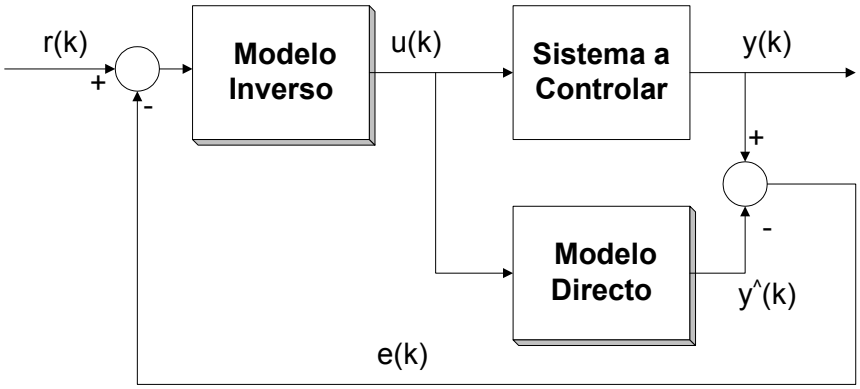


Figura 5.11: Estrutura clássica de um Controlador Baseado em Modelo Interno. O sinal $\hat{y}$(k) é a estimativa do sinal de saída gerada pelo modelo directo do sistema e e(k) o erro entre a saída do sistema e a estimativa $\hat{y}$.

do modelo inverso e M o comportamento do modelo directo e usando a regra de Mason é possível escrever a expressão seguinte:

$$H(s) = \frac{S(s).M^{-1}(s)}{1 - M^{-1}(s).M(s) + S(s).M^{-1}(s)} \tag{5.3}$$

onde $H(s)$ representa o comportamento da malha fechada no domínio de Laplace.

Como $Y(s) = H(s).R(s)$, se os modelos directo e inverso estiverem bem emparelhados de forma a que $M^{-1}(s).M(s) = 1$ então $H(s)$ reduz-se a um, fazendo com que $Y(s) = R(s)$. A conclusão que se pode extrair é que o correcto emparelhamento dos modelos directo e inverso é o suficiente para garantir a qualidade do controlo com a estrutura IMC. É também possível mostrar que este tipo de malha de controlo reduz o efeito das perturbações [89].

5.5.1 Controlador Baseado em Modelo Interno com modelos neuronais

Como é proposto em [86] a estrutura do IMC para estar de acordo com o princípio original deverá garantir o emparelhamento dos modelos directo e inverso. De acordo com as conclusões extraídas da equação 5.3, o modelo directo e inverso necessitam encaixar correctamente, ou seja o modelo inverso deve ser a função inversa do modelo directo em vez do sistema real.

Desta forma é possível garantir $M^{-1}(s).M(s) = 1$. Nestas condições os modelos serão implementados de acordo com as seguintes equações:

Modelo Directo:

$$\hat{y}(k+1) = f[\hat{y}(k), ..., \hat{y}(k - n_y + 1), u(k - td), ..., u(k - td - n_u + 1)] \tag{5.4}$$

Modelo Inverso:

$$\hat{u}(k) = f^{-1}[r(k+1), \hat{y}(k), ..., \hat{y}(k-n_y+1), u(k-td), ..., u(k-td-n_u+1)] \quad (5.5)$$

onde n_y é o número de entradas anteriores usadas, n_u é o número de sinais de controlo anteriores usados e td é o múltiplo inteiro que mais se aproxima do tempo morto do sistema.

O modelo directo comum implementaria a seguinte equação:

$$\hat{y}(k+1) = f[y(k), ..., y(k-n_y+1), u(k-td), ..., u(k-td-n_u+1)] \quad (5.6)$$

E o modelo inverso comum:

$$\hat{u}(k) = f^{-1}[r(k+1), y(k), ..., y(k-n_y+1), u(k-td), ..., u(k-td-n_u+1)] \quad (5.7)$$

O diagrama de blocos com as alterações propostas está representado na figura 5.12 [86].

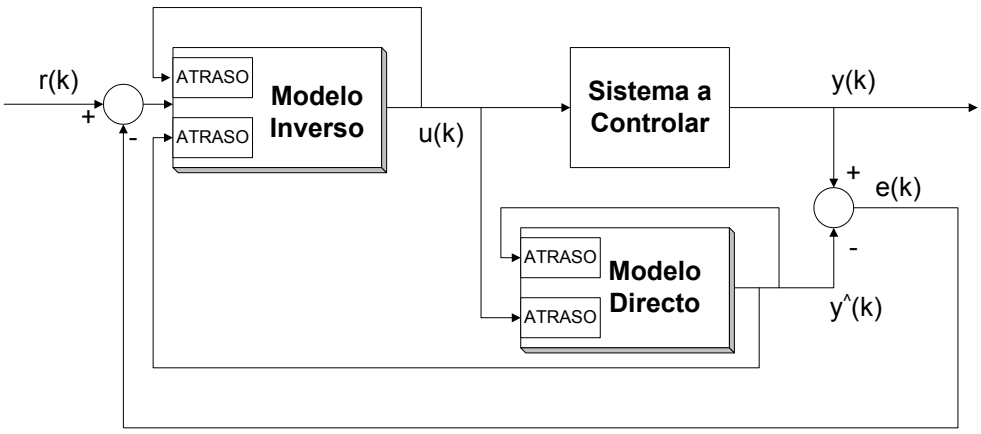


Figura 5.12: Estrutura do IMC com detalhe de implementação dos modelos directo e inverso.

Em termos práticos, nunca existe um emparelhamento total dos modelos, o que resulta em limitações da capacidade de reduzir o efeito das perturbações e na obtenção de erro em estado estacionário não nulo.

É habitual a existência de um filtro de malha entre a saída do somador e o modelo inverso, com o objectivo de melhorar a rejeição a perturbações [1].

5.6 Controlador Aditivo Baseado em Modelo Interno

A estrutura do Controlador Aditivo Baseado em Modelo Interno, apresentada pela primeira vez em [90], surge após a verificação de que a estrutura usada no Controlador Aditivo, se funcionar correctamente, equivale a um Controlador com Modelo Inverso, uma vez que após a entrada em funcionamento do modelo inverso colocado em paralelo com o controlador existente, se este modelo for de boa qualidade, levará a que o erro seja baixo e como tal a influência do controlador existente será pequena.

Como é conhecido, o Controlador com Modelo Inverso não usando directamente realimentação (poderá usá-la de alguma forma através dos regressores escolhidos) não permite obter controlo de muito boa qualidade, sendo habitualmente referenciado como dando origem a um sinal de controlo extremamente activo e que assume valores demasiado elevados [1].

Uma hipótese de melhorar um Controlador Aditivo que tem como vantagem a possibilidade de ser introduzido na malha de controlo com bastante simplicidade, consiste em inserir realimentação do erro da malha. Esta ideia conduziu à conjugação do Controlador Aditivo com o Controlador Baseado em Modelo Interno, levando à malha representada na figura 5.13.

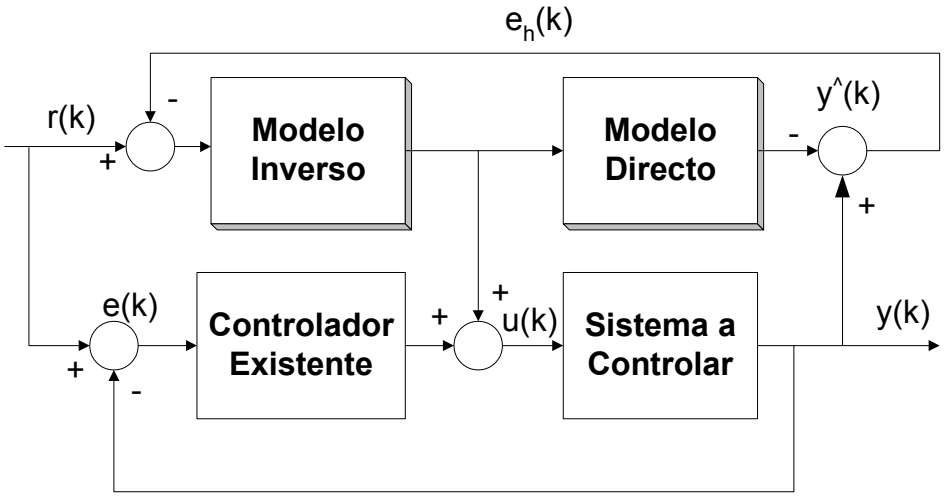


Figura 5.13: Estrutura do Controlador Aditivo Baseado em Modelo Interno.

A esta nova estrutura de controlo foi dado o nome de Controlador Aditivo Baseado em Modelo Interno (em inglês *Additive Internal Model Controller*) devido aos nomes das malhas utilizadas na sua construção.

O Controlador Existente e o Sistema a Controlar constituem a malha inicial, enquanto o Modelo Inverso, o Modelo Directo e o Sistema a Controlar compõem uma malha de Controlador Baseado em Modelo Interno. Ambas as malhas contêm realimentação, mas os sinais realimentados são diferentes. A malha inicial realimenta o

sinal de saída e tem como entrada do controlador existente o erro entre este e o sinal de referência, enquanto o Controlador Baseado em Modelo Interno realimenta o sinal de erro entre a saída do modelo directo e a saída do sistema e tem como entrada a diferença entre este e a referência.

Considerando o domínio de Laplace e a função de transferência para cada bloco (seja M e M^{-1} para os modelos directo e inverso respectivamente, - assumindo que eles estão correctamente emparelhados - S para o sistema e C para o controlador existente), a equação para o sinal $\hat{y}$ pode ser escrita da seguinte forma:

$$\hat{Y}(s) = M(s).M^{-1}(s).(R(s) - E_h(s)) \tag{5.8}$$

Supondo que, tal como foi dito, os modelos directo e inverso estão correctamente emparelhados, ou seja $M(s).M^{-1}(s) = 1$, então a equação 5.8 pode ser simplificada para:

$$\hat{Y}(s) = R(s) - E_h(s) \tag{5.9}$$

Uma vez que

$$E_h(s) = Y(s) - \hat{Y}(s) \tag{5.10}$$

A equação 5.9 pode ser reescrita da seguinte forma:

$$\hat{Y}(s) = R(s) - Y(s) + \hat{Y}(s) \tag{5.11}$$

Simplificando a equação 5.11 obtém-se:

$$Y(s) = R(s) \tag{5.12}$$

A equação 5.12 significa que a saída da malha completa seguirá a referência desde que os modelos directo e inverso estejam correctamente emparelhados.

Este resultado poderia igualmente ser obtido para a malha do Controlador Baseado em Modelo Interno, o que leva à conclusão de que esta malha funciona essencialmente como a malha do Controlador Baseado em Modelo Interno e que o Controlador Existente deixa de ser necessário, tal como acontecia no Controlador Aditivo, desde que se verifique a condição que foi colocada inicialmente.

Esta estrutura mantém as vantagens enunciadas para o Controlador Aditivo e torna ainda mais seguro, do ponto de vista da qualidade do controlo, a remoção do controlador existente, uma vez que o controlador que permanecerá em funcionamento continuará a ser um controlador de malha fechada.

Enquanto estão em funcionamento ambos os controladores o controlador existente poderá, tal como no caso do Controlador Aditivo, contribuir para a redução do sinal de erro.

A estrutura do Controlador Aditivo Baseado em Modelo Interno pode ser representada, de uma forma mais genérica, que está ilustrada na figura 5.14.

Int 1	Int 2	Int 3	Int 4	Malha Correspondente
Deslig.	Lig.	Deslig.	Deslig.	DIC
Deslig.	Lig.	Lig.	Lig.	IMC
Lig.	Lig.	Deslig.	Deslig.	AFC
Lig.	Deslig.	Deslig.	Deslig.	Malha inicial
Lig.	Lig.	Lig.	Lig.	AIMC

Tabela 5.1: Modos de funcionamento da malha genérica AIMC

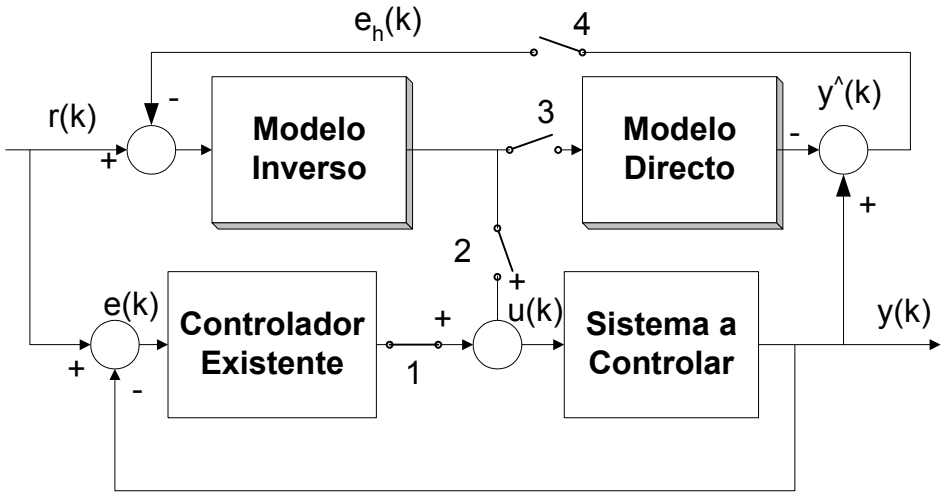


Figura 5.14: AIMC representado de forma genérica.

Neste caso está representada com quatro interruptores que servem para ilustrar os diferentes modos de operação. A tabela 5.1 apresenta o resumo dos modos de funcionamento da malha genérica do AIMC.

Como é possível verificar as estruturas de DIC, IMC, AFC e a estrutura específica do AIMC são casos particulares da estrutura genérica do AIMC, podendo ser obtidas a partir desta apenas pela escolha do estado dos interruptores.

A estrutura genérica do AIMC encontra ainda utilidade em especial quando utilizada para situações de treino *on-line*, como poderá ser visto no capítulo 7, em que poderá ser usado um controlador inicial para manter o sistema em funcionamento enquanto são recolhidos dados e preparados os modelos neuronais que serão utilizados para substituir o controlador existente ou em conjunto com ele.

Tal como acontece com o Controlador Aditivo, também a malha de AIMC deve ser implementada com algum cuidado devido à possibilidade de coexistência de controladores afectados por um factor de escala com outros que funcionam directamente com valores reais.

5.7 Conclusão

Neste capítulo foram apresentados de uma forma breve as malhas de controlo utilizadas neste trabalho ou com relação directa com estas: Controlador com Modelo Inverso, *Controller Output Error Method*, Controlador Aditivo, Controlador Baseado em Modelo Interno e introduzida uma nova estrutura designada por Controlador Aditivo Baseado em Modelo Interno que resulta de uma combinação das estruturas do Controlador Aditivo e do Controlador Baseado em Modelo Interno. Esta estrutura é a estrutura genérica da qual as outras estruturas são casos particulares.

Os resultados experimentais obtidos com estas estruturas serão posteriormente apresentados e comparados.

Capítulo 6

Um caso prático

"The aim of science is not to open the door to infinite wisdom, but to set a limit to infinite error." - Bertolt Brecht, em The Life of Galileo, poeta e dramaturgo. (1898 - 1956)

6.1 Introdução

Neste capítulo são estudadas as características estáticas e dinâmicas de um sistema real. Este sistema é utilizado para testar diversos métodos de modelização.

É feita a análise das características do sistema, da electrónica utilizada para formar a malha de identificação e controlo e apresentado o código base de acesso ao *hardware*. São também apresentadas as opções tomadas para produzir modelos e um procedimento automatizado para criar modelos optimizados e obter os valores dos parâmetros para a aplicação da Paragem de Treino Antecipada e da Regularização.

6.2 O sistema

O sistema que constitui o objecto de estudo deste trabalho é constituído por um protótipo de escala reduzida de um forno, por electrónica de condicionamento de sinal e conversão de dados, por um equipamento de arrefecimento e por um *Data Logger*.

Um esquema representativo dos elementos que compõem o forno pode ser visto na figura 6.1.

O forno é composto por uma caixa metálica exterior de chapa de aço, cheia de lã de vidro até à câmara do forno, que é delimitada por um tubo de alumina. O tubo que limita a câmara está fechado pelas flanges metálicas e pelos *o-rings* e dentro dele estão a bomba electroquímica de oxigénio e o sensor de oxigénio. No exterior estão o elemento de aquecimento e o sensor de temperatura.

Na figura 6.2 encontra-se uma vista externa do forno. Como se pode verificar, o forno é completamente fechado. A zona de operação pretendida é em torno de 750ºC, existindo um limite superior de 1000ºC. A resolução/precisão pretendida no controlo

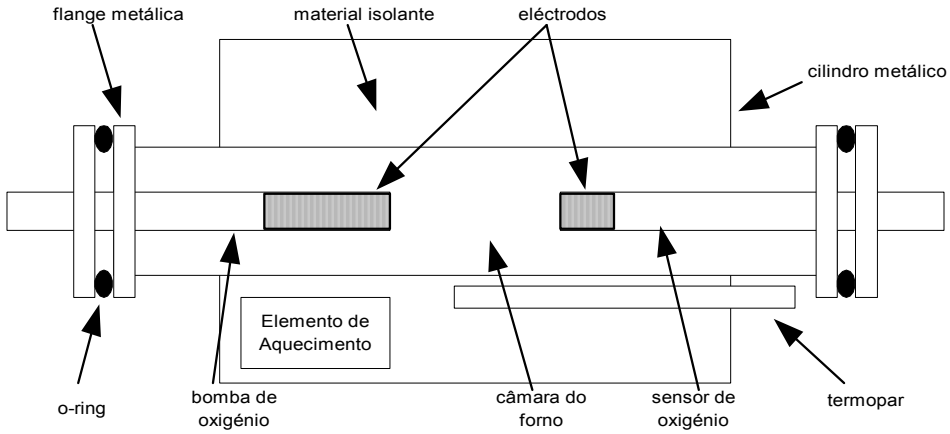


Figura 6.1: Vista esquemática do forno.

da temperatura deve ser igual ou inferior a 1^oC na gama dos 700 aos 800^oC. Adicionalmente pretende-se uma subida controlada da temperatura a partir dos 300°C. A escolha deste valor prende-se com as características do termopar B (*Rhodium-Platinum*) utilizado como sensor de temperatura.

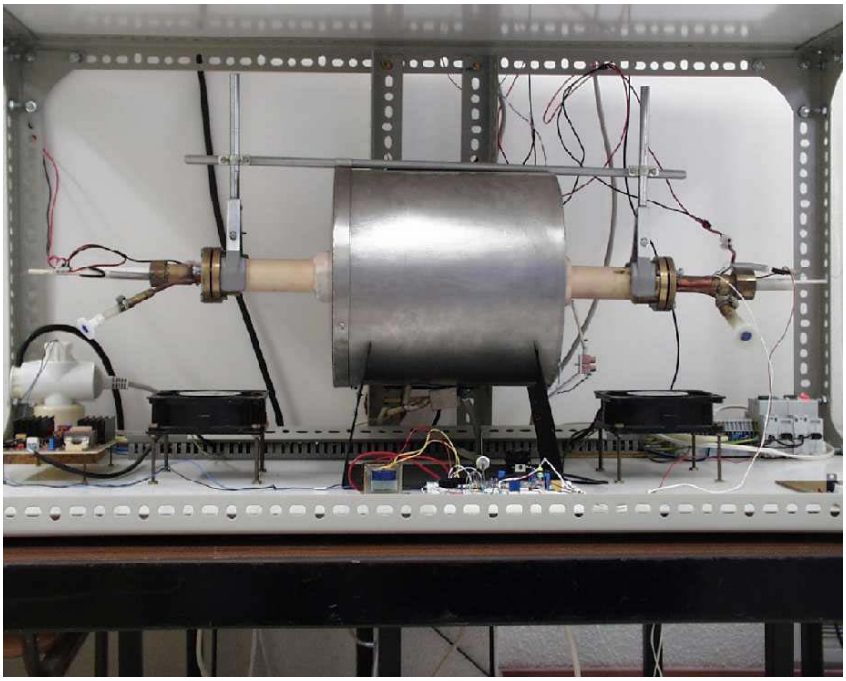


Figura 6.2: Vista exterior do forno.

O forno é aquecido por uma resistência eléctrica controlada por um módulo de potência que recebe a informação do controlador (implementado num computador tipo

PC, usando o MATLAB) através do *Data Logger* HP34970A (DL) da Hewlett Packard. Uma vez que o DL pode ser programado usando o protocolo *Standard Commands for Programmable Instruments* (SCPI), foi desenvolvido um conjunto de funções para permitir a comunicação com o programa MATLAB através da porta RS-232C. Usando os módulos HP34902A e HP34907A em conjunto com as funções desenvolvidas, é possível ler e escrever valores, analógicos ou digitais, a partir do programa MATLAB. O primeiro módulo fornece 16 entradas analógicas e o segundo fornece entradas e saídas digitais e dois conversores de sinal digital para analógico, tendo também entradas em tensão e em corrente e uma entrada específica para o termopar [91].

O diagrama de blocos da figura 6.3 representa a ligação entre os diferentes elementos que completam a malha de identificação e controlo.

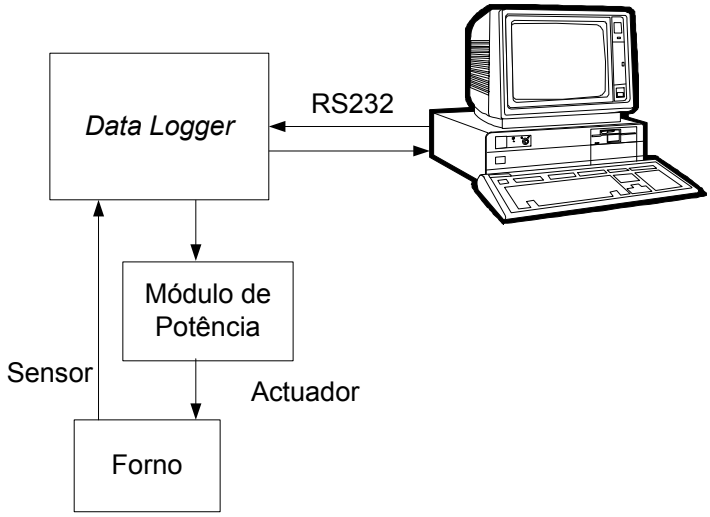


Figura 6.3: Diagrama de blocos da malha de identificação e controlo.

Na figura 6.4 pode ser vista uma fotografia do *Data Logger*.

A malha de identificação e controlo é constituída por: computador pessoal (PC), DL, módulo de potência e forno. O controlador fornece um sinal cuja gama de variação é de 0 a 4,095 V, devido às restrições impostas pelo circuito electrónico do módulo de potência, que é enviado via porta série RS232 ao DL e é em seguida convertido pelo módulo de potência. O PC utilizado ao longo deste trabalho tem um processador Celeron a 466MHz e 128MB de memória.

6.2.1 Módulo de potência

O módulo de potência converte o sinal referido anteriormente num sinal eléctrico responsável pelo aquecimento do forno através da sua resistência eléctrica. O sinal de saída do módulo de potência é um sinal de tensão alternada de valor eficaz 220V, aplicado durante um período de tempo proporcional ao sinal de entrada. A forma de processar a conversão do sinal de entrada está ilustrada na figura 6.5.

Figura 6.4: Vista do *Data Logger*.

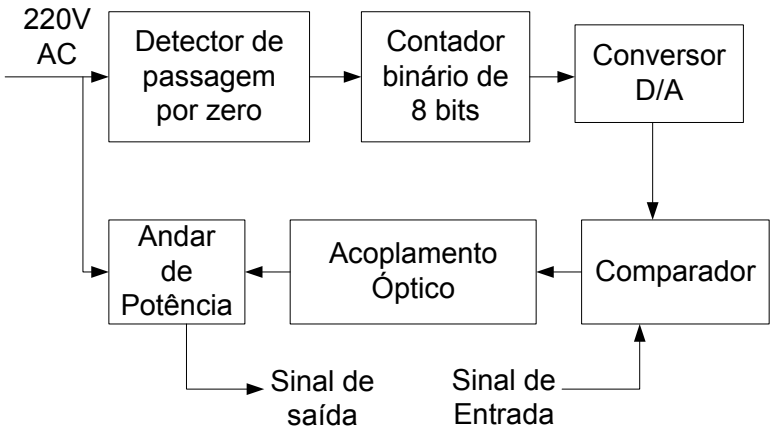


Figura 6.5: Diagrama de blocos do módulo de potência.

Um conjunto de três módulos (detector de passagem por zero, contador binário de 8 bits e conversor digital-analógico) geram um sinal em dente de serra que é comparado com o sinal de entrada dando origem a um sinal do tipo *Pulse Width Modulation* (PWM). Este sinal é aplicado ao andar de potência que colocará na saída 220V de tensão alternada nos períodos correspondentes ao sinal de entrada ter um valor superior ao sinal de dente de serra. A tensão aplicada à resistência de aquecimento é descontínua mas, como poderá ser concluído do presente trabalho, esta situação não constitui um problema, uma vez que o forno tem um comportamento do tipo passa-baixo.

A implementação física do controlador pode ser vista na figura 6.6.

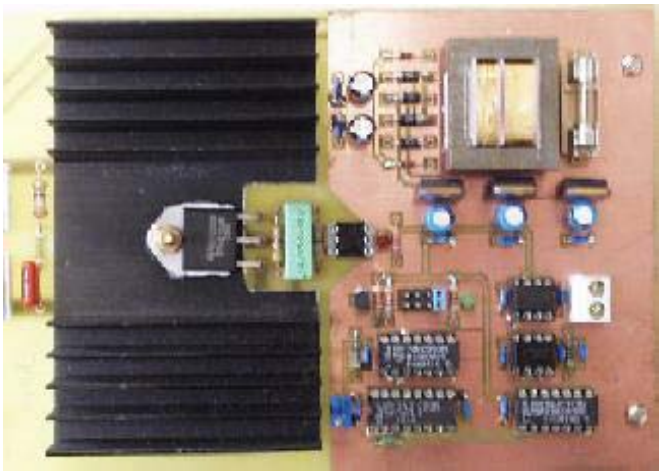


Figura 6.6: Implementação física do módulo de potência.

Controlo da potência de aquecimento

Por razões de segurança foi estabelecido que a potência máxima não deveria ultrapassar os $1000W$. Para este efeito foi incluído um disjuntor de 6 A em série com o elemento de aquecimento que é constituído por uma resistência de 30Ω.

Assim temos:

$$P_{\max} = R.I^2 = 1080W$$

Esta protecção implica que o valor máximo (prático) do sinal de entrada do módulo de potência seja ligeiramente inferior a $2800mV$.

6.3 Escolha do ambiente MATLAB

O ambiente MATLAB foi escolhido para a realização deste trabalho pela existência de *Toolboxes* com ferramentas disponíveis para a identificação e controlo, pela facilidade de acesso ao *hardware* e pelas suas capacidades gráficas.

As *Toolboxes Neural Network Based Control System Design Toolkit for use with* MATLAB [92] e *Neural Network Based System Identification Toolbox for use with* MATLAB [24], que foram utilizadas no presente trabalho, contêm diversas funções que vão desde a implementação de algoritmos, como o *Steepest descent* ou o Levenberg-Marquardt, até às funções de validação dos modelos.

O acesso ao *hardware* era uma necessidade constatada desde o início para estabelecer a malha de identificação e controlo. Com a utilização do MATLAB e o desenvolvimento das funções SCPI descritas neste capítulo foi possível concretizar esta malha.

As capacidades gráficas são importantes para rapidamente verificar a qualidade dos dados recolhidos ou do controlo efectuado.

6.4 Norma SCPI

O DL Hewlett-Packard HP3497A possui uma interface remota cujo protocolo obedece à norma SCPI. Esta norma pode ser implementada sobre suporte físico RS-232C ou HPIB (IEEE-488). Por questões de simplicidade de programação (e de interligação) optou-se pela primeira solução.

6.4.1 Função SCPI

Uma vez que se pretende utilizar o *software* MATLAB no PC foi desenvolvida a função `scpi` (sob a forma de um ficheiro CMEX[93]) capaz de dotar o programa MATLAB de capacidade de comunicação via RS-232C com o *data-logger*. Não foi possível, até à data, encontrar uma especificação formal da linguagem SCPI. A Hewlett-Packard refere-se à SCPI como uma evolução da linguagem TMSL (*Test and Measurement Systems Language*) mas não fornece (pelo menos na *internet*) pormenores significativos. Face a este panorama recorreu-se apenas à informação presente no manual do DL HP34970A [91]. Para facilitar a compreensão do trabalho efectuado são apresentadas secções do código desenvolvido.

A primeira secção apresentada é o cabeçalho da função desenvolvida, o que corresponde à ajuda disponibilizada sobre a função.

```
%
% SCPI.M
%
% SCPI Function Help File
%
% MATLAB 4.2c compatible
%
% Calling syntax:  answer = scpi(question, port_base, echo)
%
% question :  SCPI command
% port_base :  Serial Port Base Address
% echo :  0 - No echo, 1 - Echo.
%
% Standard Commands for Programmable Instruments MATLAB Function
```

Inicialização da porta série RS232C

Uma vez que a comunicação entre o PC e o *Data Logger* é efectuada sobre suporte físico RS-232C é necessário programar uma das portas de comunicação série do PC com os parâmetros adequados. Para o efeito utiliza-se a função `initrsc` (sob a forma

de um ficheiro CMEX) cuja listagem do ficheiro de ajuda se pode observar abaixo. O *Data Logger* é programado a partir do painel frontal[1].

```
%
% INITRSC.M
%
% INITRSC Function Help File
%
% RS232C COM Port Init
%
% Matlab 4.2c1 compatible
%
% Returns the PortBase Address
%
% Calling syntax:
%
% port_base = initrsc(port, baud_rate, data_bits, stop_bits, parity)
%
% port:  1, 2, 3 or 4
% baud_rate:  300,600,1200,2400,4800,9600,19200,38400,56700 or 115200
% data_bits:  7 or 8
% stop_bits:  1 or 2
% parity:  'n' (none), 'e' (even) or 'o' (odd)
```

Por questões de simplicidade não se implementa controlo de fluxo na comunicação série. Esta opção conduz, para evitar a perda de tramas, à utilização de um *baud rate* igual a 9600bps ou 19200bps. Este procedimento vai de encontro às recomendações expressas no manual do *Data Logger* e não constitui uma limitação para o trabalho.

Verificação da Presença do *Data Logger*

Após a inicialização da porta série RS232C do PC torna-se indispensável verificar se o *Data Logger* está presente e em funcionamento. A listagem abaixo permite efectuar estas duas operações.

```
% Serial COM Init
COM = initrsc(2,9600,8,1,'n');
ECHO_ON = 1;
ECHO_OFF = 0;
scpi_ans = scpi('*IDN?',COM, ECHO_ON); % Check HP34970A presence
if length(scpi_ans) ~= 0
 disp('OK - HP34970A On-Line')
```

[1] Ver os pormenores de programação no manual *HP34970A Data Aquisition/Switch Unit User's Guide*: *To configure the Remote Interface (*páginas 46-47), *Remote Interface Configuration* (páginas 150-154) e *RS-232 Interface Configuration* (páginas 270-273).

```
else
 disp('ERROR - HP34970A not present')
end
```

Leitura da Temperatura

O *Data Logger* permite a leitura directa de temperatura com base em termopares do tipo B. Abaixo é possível encontrar a sequência de comandos SCPI que o MATLAB tem de enviar para o HP3497A para configurar um canal (no caso presente o canal 105) para medir temperatura com um termopar B.

```
%
% Temperature Measurement Configuration
%
% Select the temperature measurement units
%
scpi('UNIT:TEMP C, (@105)', COM, ECHO_OFF);
disp('Kiln Temperature Measurement Units - Degrees Centigrades')
%
% Select the type of temperature transducer
%
scpi('SENS:TEMP:TRAN:TYPE TC , (@105)', COM, ECHO_OFF);
disp('Kiln Temperature Sensor - T/C')
%
% Set the integration time
%
scpi('SENS:TEMP:NPLC 1 , (@105)', COM, ECHO_OFF);
disp('Integration Time - 1 S')
%
% Select the thermocouple type
%
scpi('SENS:TEMP:TRAN:TC:TYPE B , (@105)', COM, ECHO_OFF);
disp('Kiln Temperature Thermocouple - B ')
%
% Select reference junction temperature
%
scpi('SENS:TEMP:TRAN:TC:RJUN:TYPE INT, (@105)', COM, ECHO_OFF);
disp('Kiln Reference Junction Temperature - Internal')
%
% Enable thermocouple check
%
scpi('SENS:TEMP:TRAN:TC:CHEC ON, (@105)', COM, ECHO_OFF);
disp('Kiln Thermocouple Check - ON')
```

Uma vez configurado o canal é possível ler a temperatura utilizando o seguinte código:

```
scpi_ans = scpi('MEAS:TEMP? TC, B, (@105)', COM, ECHO_ON);
kiln_temp(i) = sscanf(scpi_ans,'%f');
```

A primeira linha é responsável pelo despoletar da medida no *Data Logger* e recebe uma trama ASCII com a resposta. A segunda linha converte a trama recebida num valor de vírgula flutuante correspondente à temperatura. O índice i corresponde ao número da amostra e (consequentemente) ao tempo discreto.

Devido às características do termopar B não é possível medir (com precisão) temperaturas "baixas" (tipicamente inferiores a 250^oC). Nestas condições o HP3497A envia uma trama de *overload* (no painel do instrumento surge a mensagem +OVLD ou -OVLD) do tipo ±9,90000000E+37.

Este pormenor de funcionamento obriga a alguns cuidados extra envolvendo o teste do aparecimento deste tipo de tramas. Esta situação é inevitável durante um "arranque a frio" do forno.

Optou-se por um "arranque a frio" aplicando uma potência ao forno igual a 25% do valor máximo e esperando que a temperatura atinja os 300^oC.

```
%
% Start heating the kiln at 25 % of full power
%
scpi('SOUR:VOLT 1, (@204)', COM, ECHO_OFF);
% We must wait until the kiln is hot enough because of the B
% thermocouple characteristics.
% 300 degrees is ok !
scpi('SOUR:VOLT 2, (@204)', COM, ECHO_OFF);
init_temp = 0;
while init_temp < 300
 scpi_ans = scpi('MEAS:TEMP? TC,B, (@105)', COM, ECHO_ON);
 init_temp = sscanf(scpi_ans,'%f');
 if init_temp < 0 | init_temp > MAX_TEMP
  init_temp = 0;
 end
end
```

Considerou-se no código apresentado acima MAX_TEMP=1000ºC.

Verificou-se ainda a ocorrência esporádica de erros nas tramas de resposta do comando `scpi`. Uma vez que não foi possível detectar (e eliminar) a sua origem optou-se pela utilização de um pequeno teste do tipo:

```
%
% Read Temperature from HP3497A
%
```

```
scpi_ans = scpi('MEAS:TEMP? TC, B, (@105)', COM, ECHO_ON);
kiln_temp(i) = sscanf(scpi_ans,'%f');
%
% Take some extra precautions ...
%
if kiln_temp(i) < 0 | kiln_temp(i) > MAX_TEMP
 if i > 1
 kiln_temp(i) = kiln_temp(i-1);
 else
 kiln_temp(i) = 0;
 end
end
```

Através deste procedimento simples foi possível eliminar o aparecimento de situações em que o valor recebido correspondente à temperatura não tinha significado.

É possível que a ocorrência deste tipo de erro seja devida ao próprio funcionamento do sistema operativo Windows 98 (acesso ao *hardware*) e ao modo como foram desenvolvidas as funções do MATLAB de acesso à porta série. Uma das situações de erro mais comuns é o "desaparecimento" da letra E (da notação científica) na trama recebida. Outra situação de erro que foi identificada corresponde à recepção de um vector em vez de um único valor resultante da leitura, tendo sido colocada uma condição de verificação do tamanho da resposta e repetida a leitura no caso de ser necessário.

Envio de dados para os conversores D/A

O HP3497A possui, quando dotado do módulo HP34907A, duas saídas analógicas de tensão (designadas por DAC1 e DAC2) cuja resolução é igual a 16 bits e cuja escala é ± 12 volts. Estas duas saídas podem ser impostas através de comandos SCPI[2]. A linha apresentada abaixo coloca na saída da DAC do canal 204 a tensão de 1 volt.

```
scpi('SOUR:VOLT 1, (@204)', COM, ECHO_OFF);
```

Na situação de se pretender enviar para a DAC do canal 204 um sinal armazenado numa variável do MATLAB (que como é conhecido guarda as variáveis em formato de vírgula flutuante) basta utilizar um conjunto de comandos semelhantes ao listado abaixo.

```
%
% Send data to HP D/A Converter
%
tmp = sprintf('%2.3f', matlab_var);
scpi_data = sprintf('%c','SOUR:VOLT ', tmp, ' , (@204)' );
scpi(scpi_data, COM, ECHO_OFF);
```

[2]Ver os pormenores de programação no manual *HP34970A Data Aquisition/Switch Unit User's Guide*: *DAC Output Operations (*página 139).

No caso presente é o conteúdo da variável `matlab_var` que é enviado para o conversor A/D do canal 204.

Tempos envolvidos

As tabelas seguintes apresentam os tempos (valores aproximados) envolvidos nas diversas operações com SCPI mencionadas acima. Estes valores foram obtidos utilizando, tal como foi referido anteriormente, a linha de comunicação série sem controlo de fluxo e com *baud rates* de 9600bps e 19200bps.

Operação	Tempo (ms)
Read Temperature from HP3497A	135
Read voltage from HP3497A	134
Send data to HP D/A Converter	30

Tabela 6.1: Operações com SCPI - Tempos Envolvidos - 9600*bps*

Operação	Tempo (ms)
Read Temperature from HP3497A	119
Read voltage from HP3497A	119
Send data to HP D/A Converter	17

Tabela 6.2: Operações com SCPI - Tempos Envolvidos - 19200*bps*

Da análise conjunta das tabelas é imediato concluir que, em termos absolutos e face ao presente projecto, os tempos envolvidos são "baixos" e não diferem muito em função da velocidade de comunicação na linha série. É evidente que os mesmos devem ser, sempre que tal se justifique, tomados em consideração quando da imposição do intervalo de amostragem.

Imposição do intervalo de amostragem

No caso presente pretende-se que o algoritmo de controlo da temperatura seja executado dentro do ambiente MATLAB. Esta particularidade obriga a que seja necessário (ou pelo menos aconselhável e/ou conveniente) impor o intervalo de amostragem do algoritmo com base em funções do próprio MATLAB.

O programa MATLAB possui várias funções relacionadas com a medida do tempo. De entre várias soluções possíveis optou-se pela utilização da função `etime` [94]. Esta função calcula o tempo decorrido entre duas chamadas à função `clock`. Esta última implementa um relógio, devolvendo um vector cujo formato é:

```
[year month day hour minute seconds]
```

Os cinco primeiros elementos são inteiros. De acordo com a MathWorks [94] o elemento dos segundos (`seconds`) possui vários dígitos de precisão para além do ponto decimal.

Com base no exposto é possível implementar um esquema muito simples de imposição de um intervalo de amostragem. A listagem abaixo permite a imposição de um intervalo de amostragem igual a 150 segundos.

```
SAMPLING_PERIOD = 150; % 150 seconds of sampling interval
NMAX = 10000; % Number of samples
t0 = clock;
for i = 1:NMAX
%
% Control Loop
%
 while etime(clock, t0) < SAMPLING_PERIOD
 end
 t0 = clock;
end
```

Como é evidente esta solução apresenta alguns problemas dos quais o mais importante é tratar-se de um procedimento do tipo *polling*: o programa fica "preso" na instrução `while` até que decorra o tempo previsto para o intervalo de amostragem. Face aos valores pretendidos para o intervalo de amostragem (vários segundos) e a capacidade computacional disponível, esta limitação não foi considerada muito severa para o presente projecto.

Visualização gráfica dos resultados

Uma das vantagens da utilização do MATLAB como ambiente de programação reside nas suas capacidades gráficas. Estas permitem, de um modo simples e directo, a visualização das variáveis envolvidas. No caso presente pretende-se, entre outras, a visualização das variáveis mais importantes de uma malha de controlo: sinal de referência, sinal de saída e sinal de controlo (valor absoluto e incrementos).

São, de seguida, apresentadas duas soluções para esta visualização: a primeira (mais simples) afixa em duas janelas gráficas todos os valores das variáveis; a segunda solução afixa apenas uma janela temporal das mesmas variáveis.

O código correspondente pode ser observado na listagem abaixo.

```
%
% Plot all the important data
%
figure(1)
plot([kiln_temp(1:i) set_point(1:i)]);
grid, title('Kiln Temperature & Set_Point')
xlabel('Time x 150s'), ylabel('Temperature (Centigrade)')
```

```
figure(2)
plot([u(1:i) delta_u(1:i)]);
grid, title('Control Signal & Increments of Control Signal')
xlabel('Time x 150s')
drawnow
```

É evidente que a esta solução implica tempos de execução crescentes com o número de pontos. Para as condições de teste verificou-se que o tempo necessário para a afixação dos resultados é aproximadamente igual a 0,25 segundos até aos 200 pontos crescendo (em escada) até 1,54 segundos para os 10000 pontos.

Como foi referido atrás, a outra solução passa pela utilização de uma "janela de visualização" de tamanho temporal limitado. Neste caso partiu-se do princípio de que uma janela com 100 pontos (corresponde a 100 intervalos de amostragem) seria adequado.

```
MAX_POINTS=100;
%
% Plot all the important data - Limited Window Version
%
figure(1)
plot([kiln_temp(ii:i) set_point(ii:i)]);
grid, title('Kiln Temperature & Set_Point')
xlabel('Time x 5s'), ylabel('Temperature (Centigrade)')
figure(2)
plot([u(ii:i) delta_u(ii:i)]);
grid, title('Control Signal & Increments of Control Signal')
xlabel('Time x 5s')
if i-ii > MAX_POINTS
 ii = i - MAX_POINTS;
end
drawnow
```

Esta solução proporciona um tempo de afixação das variáveis que é proporcional a `MAX_POINTS` e independente do instante de amostragem (neste caso a variável `i`). Para as condições de ensaio verificou-se que valor médio do tempo de processamento é igual a 0,027 segundos.

Solução	Tempo (ms)
Total	25 a 1540
Janela	27 (média)

Tabela 6.3: Visualização Gráfica dos Resultados - Tempos Envolvidos

Entrada (Volts)	Temperatura	
	Subida	Descida
0,3	348°C	330°C
0,4	470°C	450°C
0,5	570°C	575°C
0,6	668°C	675°C
0,7	760°C	770°C
0,8	837°C	840°C
0,9	913°C	

Tabela 6.4: Temperaturas finais obtidas nos testes em malha aberta.

Comparando os tempos obtidos é imediato concluir que a escolha de uma ou outra solução depende do intervalo de amostragem e da pertinência da visualização de toda a evolução temporal das variáveis.

6.5 Aquisição de dados

6.5.1 Escolha do período de amostragem

O período de amostragem escolhido foi de 150 segundos e os resultados que serão apresentados (com excepção do capítulo 8) correspondem a este valor de amostragem. Como poderá ser verificado pelos resultados referidos no capítulo 3 e pelos dados apresentados na secção seguinte este valor do período de amostragem leva a um número de amostras por tempo de subida ligeiramente mais elevado do que é proposto por alguns autores (ver capítulo 3), no entanto este valor foi escolhido após a obtenção de modelos e da sua validação não só através da sequência de teste mas também pela utilização dos modelos em malhas de controlo. Na fase de escolha do período de amostragem foram obtidos e testados modelos com diversos valores de período de amostragem, com destaque para os modelos de 30s, 60s e de 300s, com os quais também foi possível obter controlo de "qualidade aceitável".

6.5.2 Resposta em malha aberta

Na figura 6.7 pode ser vista a resposta em malha aberta a diversos degraus de subida e descida, utilizando um período de amostragem de 150 segundos.

A tabela 6.4 apresenta os valores finais de temperatura obtidos nas fases de subida e descida da resposta em malha aberta do forno de escala reduzida. Como é possível verificar nem sempre foram obtidos os mesmos valores finais, chegando a existir diferenças de 20°C.

Na tabela 6.5 são apresentados os tempos de subida e de descida, em termos de número de amostras.

Com os dados das tabelas 6.4 e 6.5 é possível traçar a característica estática do sistema que é apresentada na figura 6.8. Como se pode verificar, a curva de aquecimento

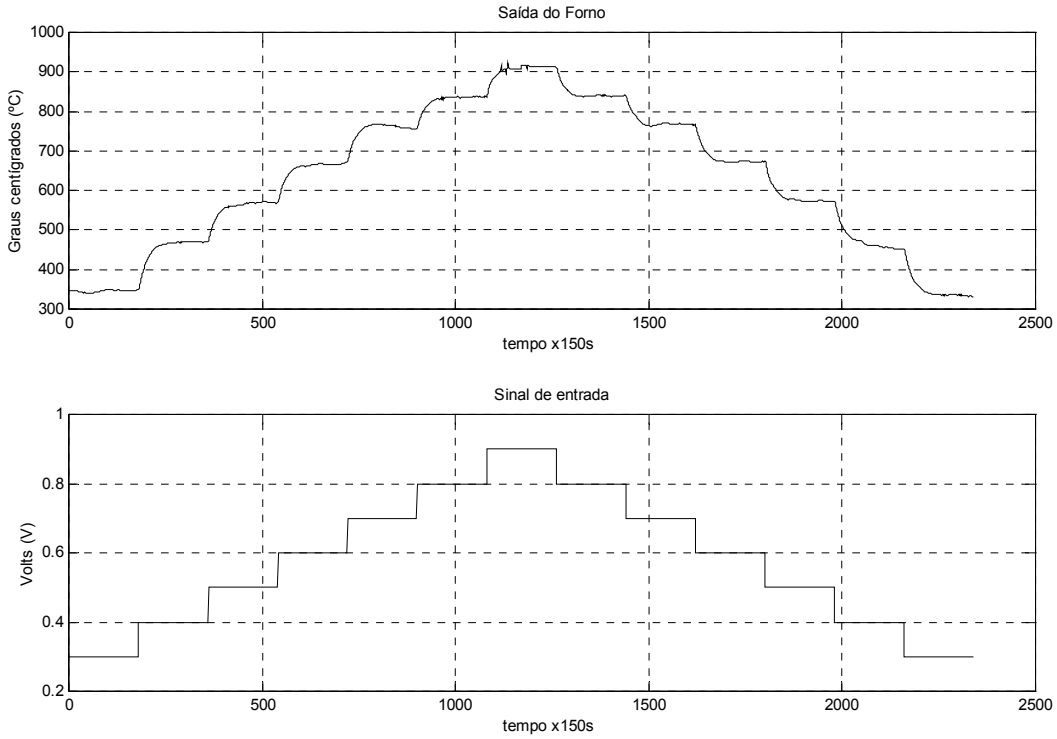


Figura 6.7: Resposta do sistema em malha aberta.

Transição (Volts)	Temperatura	
	Subida	Descida
0,3 - 0,4	47	56
0,4 - 0,5	63	83
0,5 - 0,6	48	43
0,6 - 0,7	34	37
0,7 - 0,8	43	32
0,8 - 0,9	39	32

Tabela 6.5: Subidas de temperatura resultantes dos diversos degraus de tensão na entrada.

não coincide totalmente com a curva de arrefecimento.

Estes valores apresentam variações elevadas, ao contrário do que seria de esperar para um sistema linear onde os diversos tempos de subida e de descida deveriam ser constantes. Destes resultados é possível verificar não só a não linearidade (os valores finais obtidos nas fases de subida e descida não são idênticos, os tempos de subida e descida nos vários intervalos também não são iguais, tal como os acréscimos/decréscimos de temperatura correspondentes a variações semelhantes na entrada) do sistema em causa, como a existência de ruído de medida. Este ruído de medida resulta da utili-

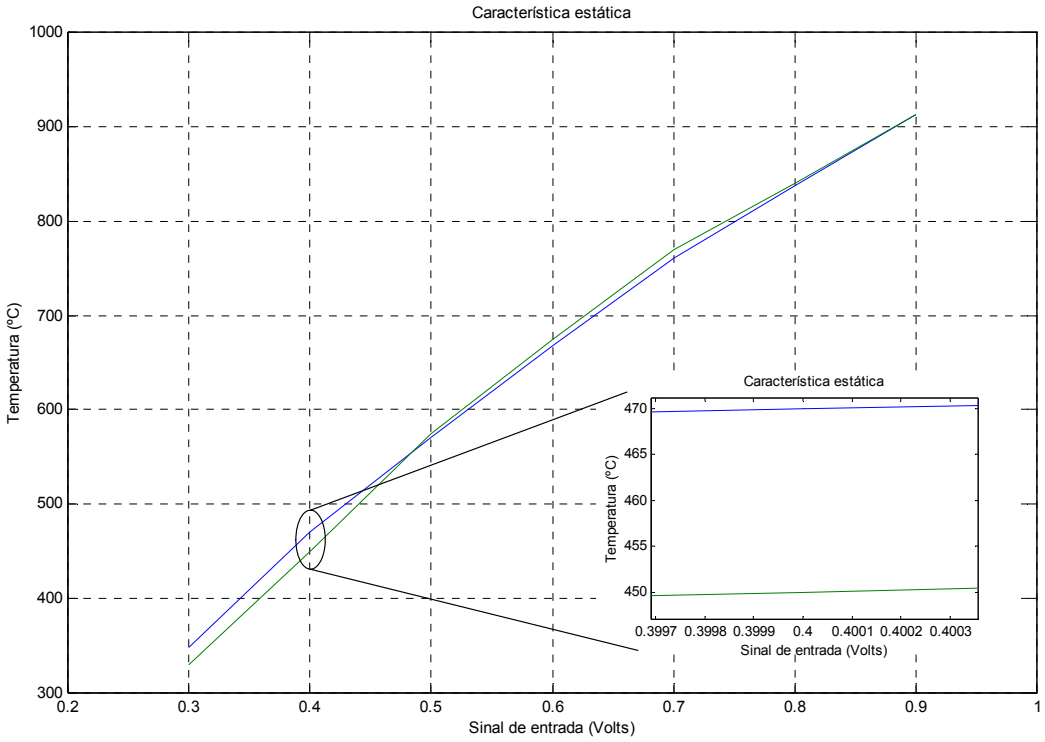


Figura 6.8: Característica estática do sistema sem coincidência total entre o aquecimento e o arrefecimento.

zação do sensor de temperatura do termopar tipo B.

6.5.3 Escolha do tipo de dados a utilizar

Os dados utilizados para a identificação foram dados de malha aberta, tendo havido preocupação em escolher um sinal de entrada que proporcione um espectro de amplitudes suficientemente largo por forma a garantir uma correcta identificação de um sistema não-linear [15] e que cubra a gama de valores que leve a saída a um estado de operação semelhante ao pretendido em funcionamento normal.

Na figura 6.9 pode ser visto o sinal utilizado, 75% dos pontos (os iniciais) foram utilizados como sequência de treino e os restantes como sequência de teste.

6.6 Preparação dos dados

Os dados foram afectados de um factor de escala, como foi referido em 3.3.1. Os dados escalados podem ser vistos na figura 6.10, com separação dos conjuntos de treino e de teste.

Como foi indicado, os dados foram separados em dois conjuntos, treino e teste. O conjunto de treino é utilizado para treinar os modelos e o de teste para validar esses

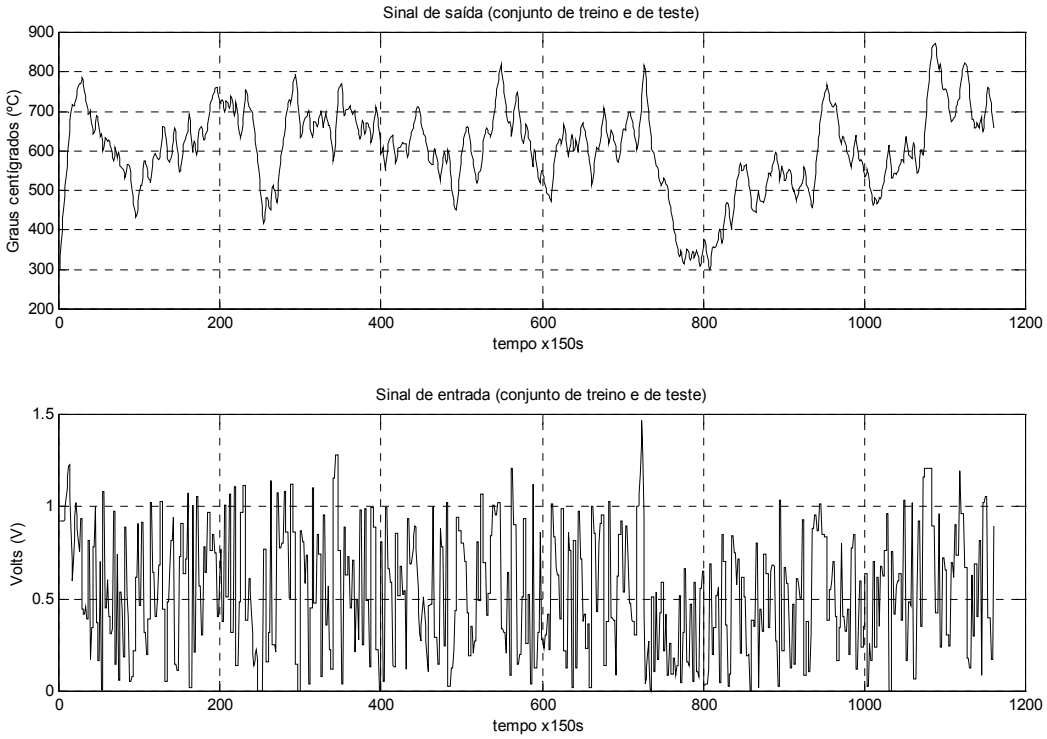


Figura 6.9: Sinal utilizado na preparação dos modelos neuronais, 75% dos pontos (os iniciais) foram utilizados como sequência de treino e os restantes como sequência de teste.

modelos. A comparação da qualidade dos modelos obtidos foi conseguida com a própria sequência de teste ou pela utilização dos modelos em malhas de controlo.

No presente trabalho não foi feito pré-processamento dos dados nem filtragem por se verificar que não havia necessidade de utilizar estas soluções.

6.7 Classes de modelos

A classe de modelos utilizada foi a classe NNARX. Como foi explicado em 3.4.2 esta é a classe mais "conveniente" para os modelos neuronais.

6.8 "Ordem" do sistema

De acordo com o explicado em 3.5, verificou-se que o sistema deve ser modelizado com dois regressores do sinal de entrada e dois regressores do sinal de saída. Apesar disso, como será documentado na secção 6.12, foi preparado um procedimento automático de criação de modelos que permite a escolha dos regressores mais "importantes" para a qualidade do modelo.

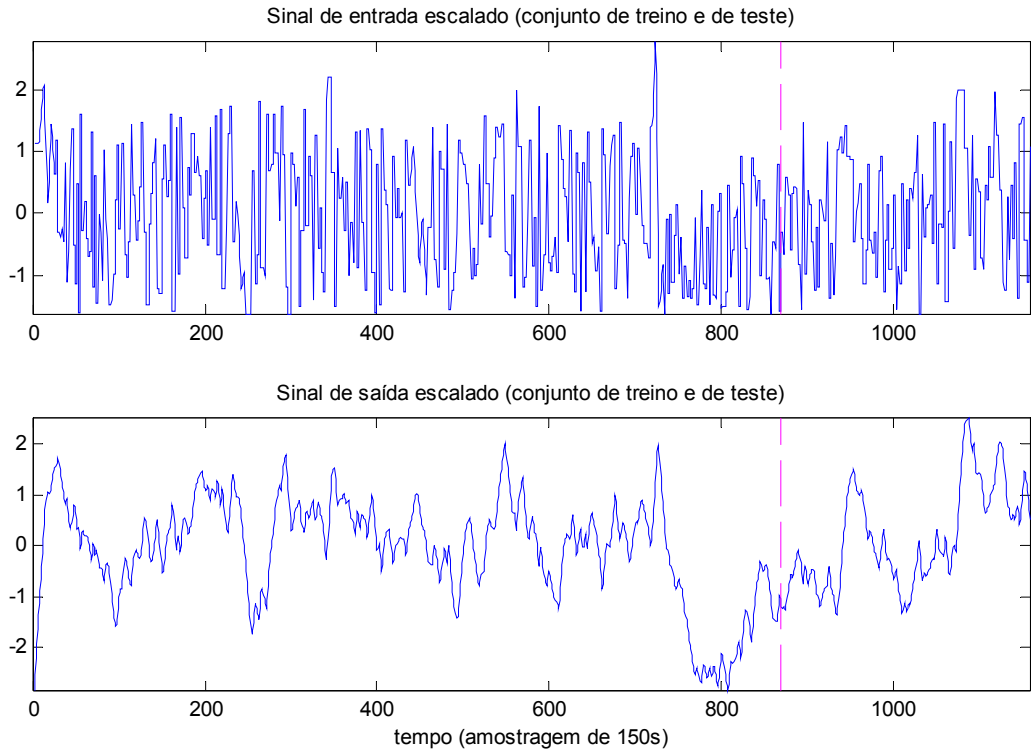


Figura 6.10: Sinal escalado utilizado na preparação dos modelos neuronais.

6.9 Estruturas de treino

Todas as estruturas de treino descritas no capítulo 3 foram testadas, no entanto, os modelos inversos que serão apresentados no capítulo 7, foram produzidos com a estrutura de treino genérica e, em alguns casos, com a técnica híbrida genérica/especializada.

6.10 Capacidade de generalizar

Para melhorar a capacidade de generalizar foram utilizadas as técnicas de Paragem de treino antecipada e Regularização devido à sua menor complexidade. Nos modelos em que os parâmetros foram optimizados com recurso ao operador humano, foi utilizada apenas a Paragem de treino antecipada e nos modelos em que foi utilizado um procedimento automatizado, foram utilizadas ambas as técnicas.

6.11 Modelos lineares

Com o objectivo de verificar se os modelos lineares ARX obtinham qualidade suficiente para serem utilizados, foram efectuados alguns testes, utilizando modelos identificados com o algoritmo dos mínimos quadrados.

A figura 6.11 mostra o resultado de simulação de um modelo linear de 1ªordem sobre a sequência de teste. Este modelo está representado na equação 6.1, onde $y(k)$ corresponde ao valor discreto da temperatura e $u(k)$ ao valor discreto do sinal de controlo.

$$y(k) = a_1.y(k-1) + b_1.u(k-1) \tag{6.1}$$

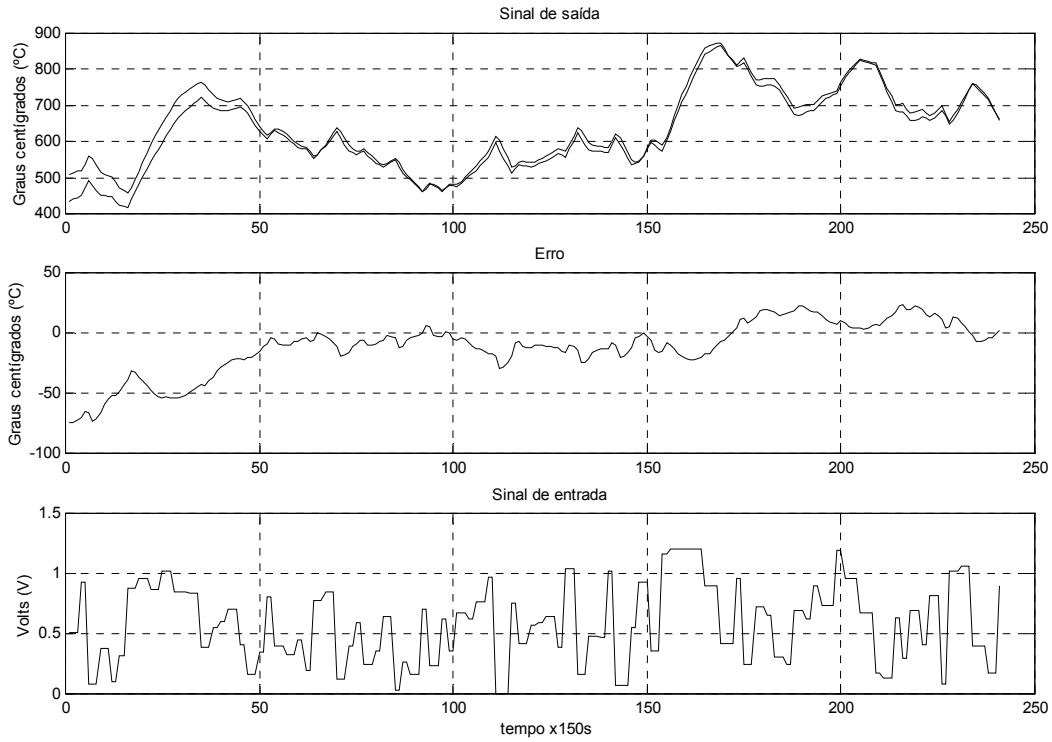


Figura 6.11: Resultados de simulação do modelo linear de 1ª ordem.

A partir da análise da figura é possível verificar que existem zonas em que a simulação se afasta, de forma considerável, do sinal de teste, apesar de terem sido retirados os 50 pontos iniciais. O valor do EQM obtido neste caso foi de 621,73, o que dá um erro por amostra de quase 25ºC. Apesar de se tratar de uma simulação da saída a partir da entrada da sequência de teste será possível comparar este resultado com os que serão apresentados no capítulo 7 e verificar que estes valores são demasiado elevados.

A figura 6.12 apresenta um resultado semelhante ao anterior, mas neste caso para um sistema de 2ªordem. Tal como no caso anterior, existem zonas em que o erro é elevado, resultando num EQM de 388,31, o que significa cerca de 19,7ºC de erro médio por amostra. Também neste caso foi concluído que o erro era demasiado elevado.

O estudo dos modelos lineares permitiu também a análise da ordem do sistema no caso de ser modelizado como um sistema linear.

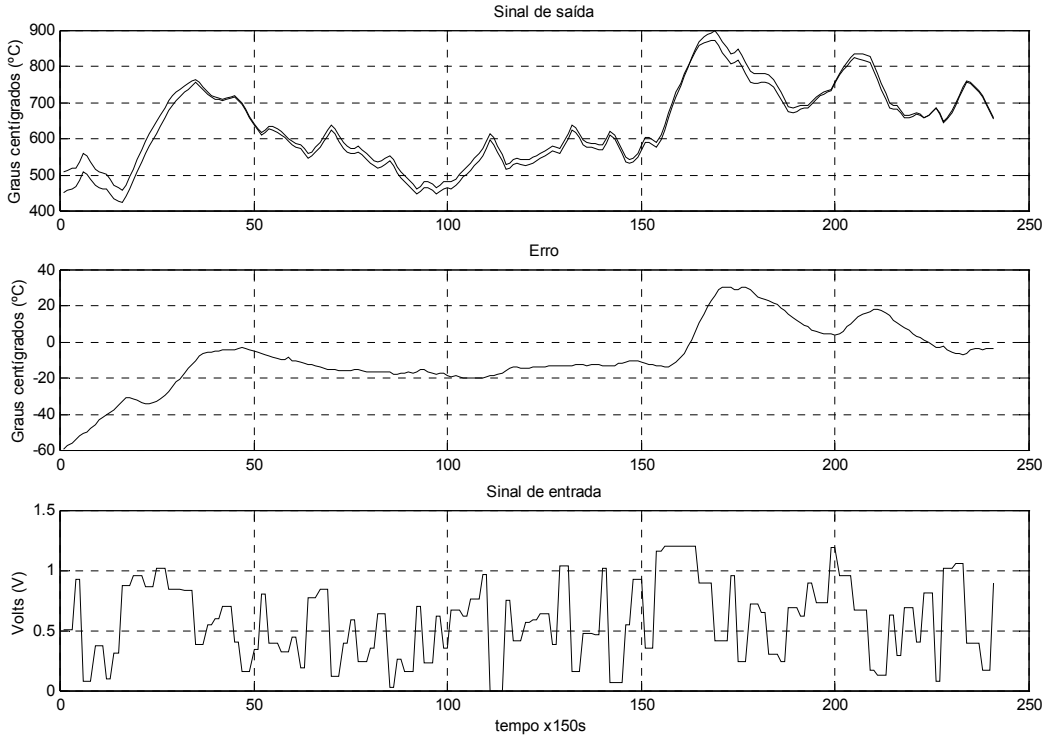


Figura 6.12: Resultados de simulação do modelo linear de 2ª ordem.

No caso de se tratar de um sistema de 1ªordem, o modelo tem um pólo em 0,9624 e no caso de ser de 2ªordem existe um zero em -0,0217 e dois pólos, um em 0,2917 e o outro em 0,955. Quando foram testados modelos de ordem superior verificou-se a existência de cancelamento pólo-zero.

6.11.1 Modelos baseados em Redes Neuronais

Estes primeiros modelos obtidos foram optimizados utilizando os conhecimentos do operador humano relativamente às técnicas de modelização e às RNs, serão designados por modelos optimizados pelo operador humano. Todos os modelos optimizados pelo operador humano, foram treinados com recurso ao algoritmo de Levenberg-Marquardt, embora como poderá ser verificado no presente documento, a utilização do algoritmo tenha sido feita com diversas variantes.

Os primeiros pares de modelos criados, optimizados com recurso aos conhecimentos do operador humano, são muito semelhantes, tendo em comum o facto de usarem dois regressores do sinal de entrada e dois regressores do sinal de saída, oito neurónios na camada escondida, uma saída linear, 256 épocas de treino, paragem de treino se o erro for nulo (em termos práticos isto significa que o treino é terminado pelo número máximo de épocas de treino, uma vez que o erro nunca será nulo), parâmetro de regularização nulo e valor inicial do parâmetro $\mu = 1$ (ver secção 2.7.1), sendo diferentes apenas nos

valores dos pesos. Este par de modelos será designado por par_1.

Estes modelos, tal como todos os modelos produzidos ao longo do trabalho apresentado nesta tese, são treinados a partir de pesos aleatórios com valores entre 0 e 1. Estes valores foram escolhidos porque as RNs vão trabalhar com entradas escaladas.

Os resultados de controlo obtidos com estes modelos serão apresentados no capítulo 7 e os resultados de simulação dos modelos directos usando a sequência de teste são apresentados na figura 6.13. São apenas apresentados os resultados obtidos para os modelos directos por uma questão de limitação de espaço.

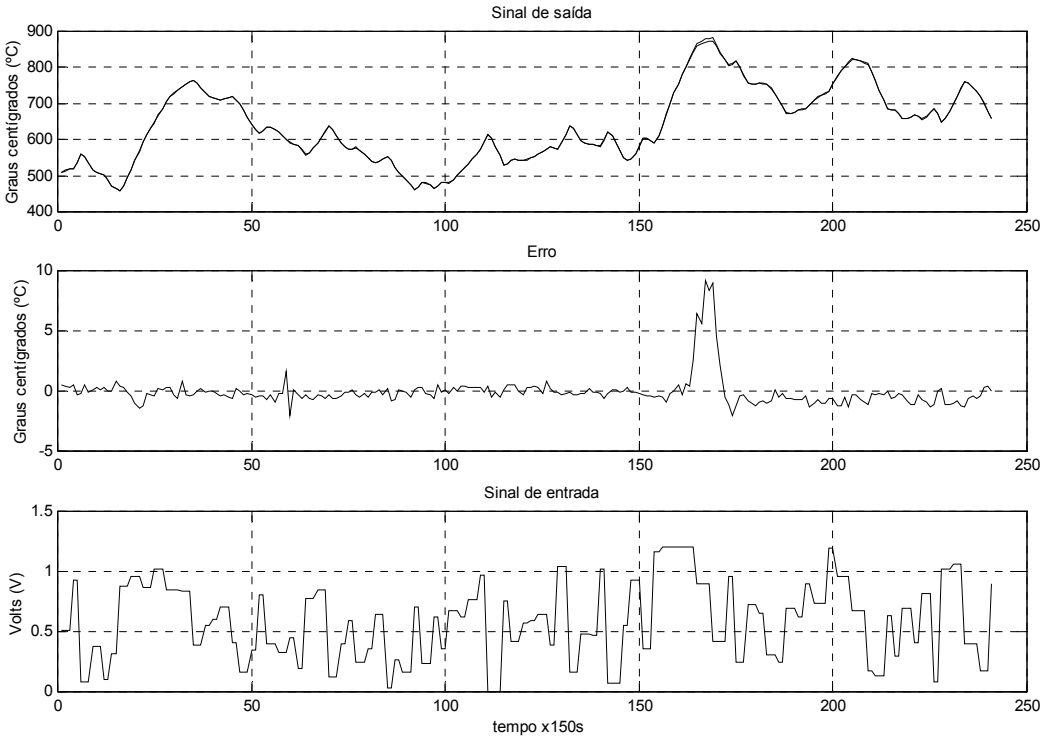


Figura 6.13: Resultados de simulação do modelo neuronal optimizado com recurso aos conhecimentos do operador humano.

Como é possível verificar, mesmo por simples observação da figura 6.13, a qualidade deste modelo é muito superior à obtida com o modelo linear, sendo o EQM de 1,75 o que dá 1,32ºC de erro por amostra, um valor muito menor do que o obtido com o modelo linear.

6.12 Automatização do processo de optimização de modelos

Depois da obtenção dos primeiros modelos, cujas características foram optimizadas pelo operador humano, foi implementado e testado um processo automático de criação

de modelos com o objectivo de melhorar a sua qualidade global. Optou-se por utilizar como ferramenta de optimização os AGs e por testar soluções com paragem de treino antecipada (Regularização Implícita), como era já feito com os modelos iniciais, e com Regularização Explícita. Parte deste trabalho consta dos artigos [55] e [95].

Nesta parte do trabalho pretendeu-se não só testar se era possível obter modelos de melhor qualidade (directos ou inversos) do que os optimizados pelo operador humano, como criar um procedimento automático que permita obter os parâmetros da RN e comparar as técnicas de Regularização Explícita e Implícita. Os parâmetros em questão são relativos à paragem de treino antecipada (número de iterações ou de épocas), à Regularização e à estrutura da RN (número de neurónios da camada escondida e número de regressores de cada entrada).

6.12.1 Introdução

O problema do treino excessivo nas RNs tem sido um tópico de trabalho desta área originando elevado número de publicações de muitos investigadores como foi documentado na secção 3.7. Sendo as técnicas mais comuns para evitar o problema de *overfitting* o *Early Stopping* e a Regularização e não existindo uma forma determinística de calcular os parâmetros usados nestas técnicas (número de iterações e *weigh decay,* respectivamente - ver capítulo 3) é necessário utilizar métodos iterativos que garantam a optimização destes valores e, consequentemente, dos modelos resultantes.

Embora estas duas técnicas tenham sido consideradas formalmente equivalentes [59] é também importante saber se em termos práticos algumas delas desempenha melhor a sua função ou é mais simples de implementar.

Para este efeito foi desenvolvido um processo automático de optimização de modelos com recurso aos AGs. Neste processo foram preparados modelos directos e inversos, estes últimos utilizando a técnica híbrida genérica/especializada para modelos inversos, que serão comparados no capítulo 7 através da utilização dos modelos em malhas de controlo.

Esta solução não é do tipo caixa preta (do inglês *black box*), uma vez que é utilizado algum conhecimento prévio sobre o sistema para a escolha das gamas de variação dos parâmetros. Neste caso a solução deverá ser classificada como caixa cinzenta (do inglês *grey box*).

6.12.2 Detalhes de implementação

O processo de escolha dos modelos directos e inversos é baseado no mesmo princípio, embora existam algumas diferenças entre as duas implementações. Existe um procedimento comum que utiliza AGs para gerar cada solução em função das combinações possíveis na estruturas das RNs e diferentes formas de avaliar a aptidão de cada indivíduo.

A estrutura da rede é composta por quatro parâmetros: número de entradas passadas, número de saídas passadas, número de neurónios na camada escondida e número

Parâmetro	Nº de bits	Gama
Saídas Anteriores	2	1:4
Entradas Anteriores	2	1:4
Neurónios	5	1:32
Iterações/ *weigh decay*	11	1:2048 1E-5:1E3

Tabela 6.6: Número de bits e gamas de variação permitidas para os parâmetros.

de iterações de treino ou valor do parâmetro de regularização. Os primeiros dois parâmetros permitem escolher a informação a ser utilizada pelos modelos e diferenciar a importância de entradas e saídas anteriores, utilizando apenas o que é importante.

O número de neurónios na camada escondida permite o ajuste do tamanho da rede à complexidade do sistema a ser modelizado.

A utilização do número de épocas de treino permite a aplicação de *Early Stopping* ou em alternativa será codificado o valor do *weigh decay* para a aplicação da regularização.

Na tabela 6.6 está resumida a informação utilizada para codificar cada solução, indicando a gama de valores permitida para cada parâmetro. A escolha das gamas de variação resulta do conhecimento obtido com o trabalho prévio sobre este sistema.

Cada modelo obtido é uma implementação da estrutura definida e de um valor para cada um dos parâmetros da tabela 6.6.

A estrutura base foi fixada no início, tratando-se de RNs sem realimentação, com uma camada escondida usando como função de activação a tangente hiperbólica e saída com função de activação linear. Por uma questão de coerência entre os resultados obtidos com *Early Stopping* e Regularização, o número de iterações de treino utilizado neste último caso foi de 2048, ou seja o valor máximo possível com *Early Stopping*.

6.12.3 Optimização com AGs

No capítulo 4 foram explicados os mecanismos utilizados pelos AGs para procurar uma solução de forma iterativa. A presente implementação utiliza os operadores de cruzamento, mutação e elitismo. Estes três operadores base foram considerados indispensáveis uma vez que com o cruzamento se assegura a criação de novas gerações, com a mutação, pelo seu caracter aleatório, é garantida a possibilidade de atingir qualquer solução no universo proposto (diminuindo a dependência em relação às soluções iniciais) e o elitismo garante que a evolução não conhece retrocessos uma vez que um sub-conjunto das melhores soluções é guardado.

A implementação criada faz uso de uma população de 20 indivíduos, de um forte elitismo de 20%, de cruzamento de um só ponto e de mutação de 5% dos genes de toda a população (excepto as elites) em cada geração.

O operador de mutação é uma máscara binária gerada aleatoriamente, de acordo com a percentagem pré-definida, que é sobreposta à codificação binária da população

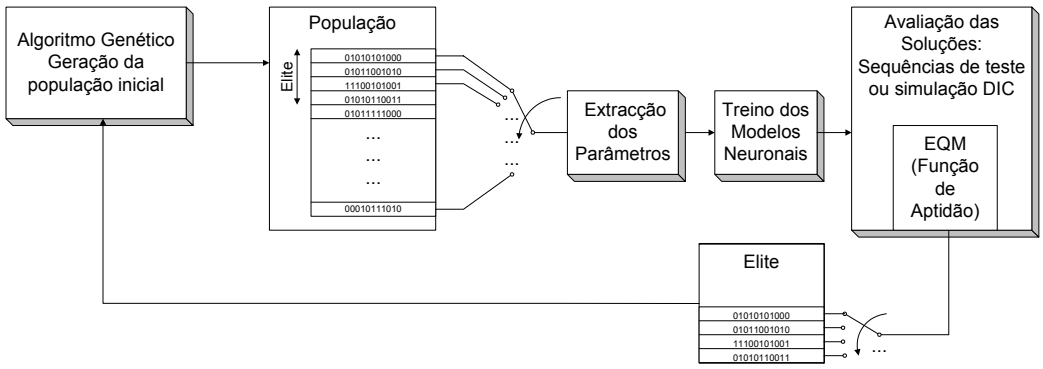


Figura 6.14: Diagrama de blocos do sistema de optimização com AGs.

modificando alguns dos bits.

O cruzamento é efectuado sobre 50% da população incluindo sempre as elites. Os indivíduos são escolhidos para reprodução de forma aleatória, com oportunidades idênticas, para criar a população da nova geração.

Como função de aptidão é utilizado o EQM entre a saída do modelo criado e a saída pretendida. Os valores pretendidos podem ser os constantes do conjunto de dados de teste, no caso do modelo directo, ou o sinal de referência usado numa simulação de DIC, no caso do modelo inverso. Naturalmente, nesta situação, a melhor solução é aquela que apresenta o menor valor da função de aptidão.

Uma perspectiva mais completa da implementação pode ser obtida da figura 6.14.

O bloco designado por AG representa a geração das soluções. É neste bloco que é gerado o conjunto de soluções iniciais e, em cada geração, um novo conjunto de soluções para completar a população.

A população é representada como uma tabela de conteúdo binário, com cada linha a representar um indivíduo da população e como tal um modelo neuronal. Os parâmetros dos modelos são depois extraídos, ou seja, transformados de sequências binárias em valores reais de forma a ser possível construir o modelo correspondente que em seguida é treinado com o algoritmo de Levenberg-Marquardt.

A avaliação das soluções é feita através da função de aptidão, que consiste na utilização de uma sequência de teste no caso do modelo directo ou de uma simulação de DIC no caso do modelo inverso, e da qual resulta um valor de EQM que permite seriar as soluções de acordo com a sua qualidade.

Após a seriação das soluções, é feito um registo da população com vista a seleccionar os melhores elementos que integrarão a elite que será utilizada pelo AG, em conjunto com outros indivíduos, para gerar os novos elementos, dando origem à próxima geração.

Este ciclo é repetido até ser atingida uma das condições de paragem definidas: número máximo de gerações (definido como 1000), número máximo de gerações sem alteração em nenhum elemento da elite (definido como 50) ou valor mínimo de erro (definido como 0), sendo habitualmente a 2ª condição a responsável pela paragem.

Os modelos resultantes deste procedimento automático são designados por par_2

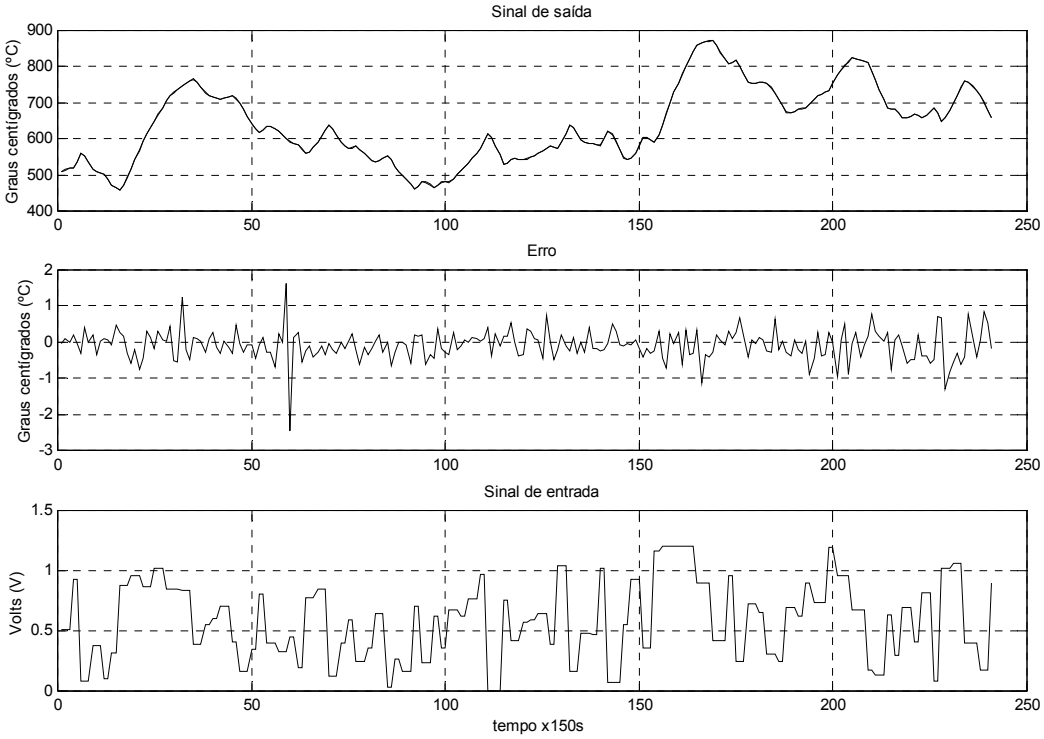


Figura 6.15: Resultados de simulação do modelo neuronal optimizado com Algoritmos Genéticos utilizando a técnica de Paragem de Treino Antecipada (par_2).

(regularização implícita) e par_3 (regularização explícita) e os respectivos resultados de controlo são apresentados no capítulo 7. O par_1 é composto pelos modelos directo_1 e inverso_1 e assim sucessivamente. Os resultados de simulação dos modelos directos usando a sequência de teste são apresentados nas figuras 6.15 e 6.16.

Para o resultado apresentado na figura 6.15 o valor do EQM é de 0.17, o que dá um erro por amostra de 0,41°C.

As características destes modelos e dos modelos iniciais podem ser vistas na tabela 6.7, onde NU é o número de entradas anteriores do sistema, NY é o número de saídas anteriores do sistema, δ é o parâmetro de regularização, N.A. significa não aplicável, Neurónios refere-se ao número de neurónios da camada escondida, o erro de teste é o erro obtido por simulação, no caso dos modelos directos e o erro de avaliação por simulação de DIC no caso dos modelos inversos (em ambos os casos trata-se de EQM sobre valores escalados, pelo que o seu interesse se resume apenas a um termo de comparação) e gerações indica o número de gerações utilizadas pelo AG até terminar a sua optimização.

Para o resultado apresentado na figura 6.16 o valor do EQM é de 0.21, o que dá um erro por amostra de 0,46°C.

As figuras 6.17, 6.18, 6.19 e 6.20 mostram a evolução dos melhores indivíduos através das gerações. As figuras 6.17 e 6.18 são referentes à utilização da técnica

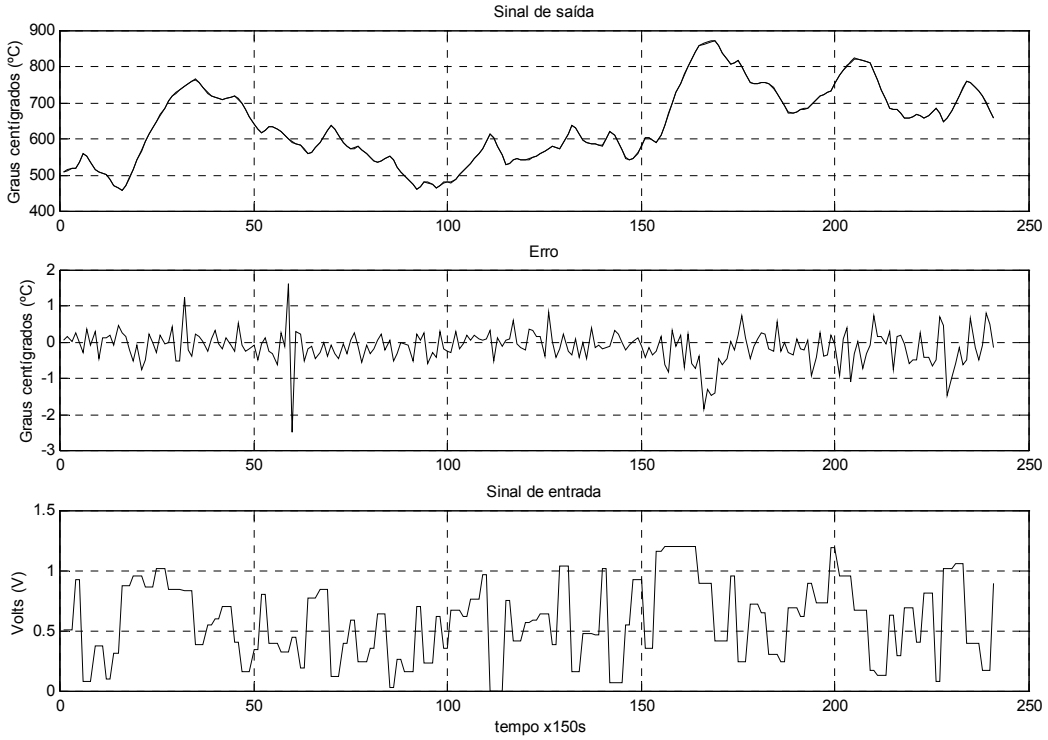


Figura 6.16: Resultados de simulação do modelo neuronal optimizado com Algoritmos Genéticos utilizando a técnica de Regularização (par_3).

Modelos/ Parâmetros	NU	NY	Neurónios	Iterações/ δ	Erro de teste	Gerações
Optimizados pelo operador humano						
Directo_1	2	2	8	256	$6{,}42e^{-5}$	N.A.
Inverso_1	2	2	8	256	$4{,}40e^{-5}$	N.A.
Early Stopping						
Directo_2	3	4	2	899	$6{,}61e^{-6}$	99
Inverso_2	4	4	4	41	$1{,}30e^{-6}$	112
Regularização						
Directo_3	4	4	2	$1e^{-5}$	$8{,}16e^{-6}$	98
Inverso_3	2	4	11	$1{,}79e^{-5}$	$7{,}46e^{-3}$	129

Tabela 6.7: Número de bits e gamas de variação permitidas para os parâmetros.

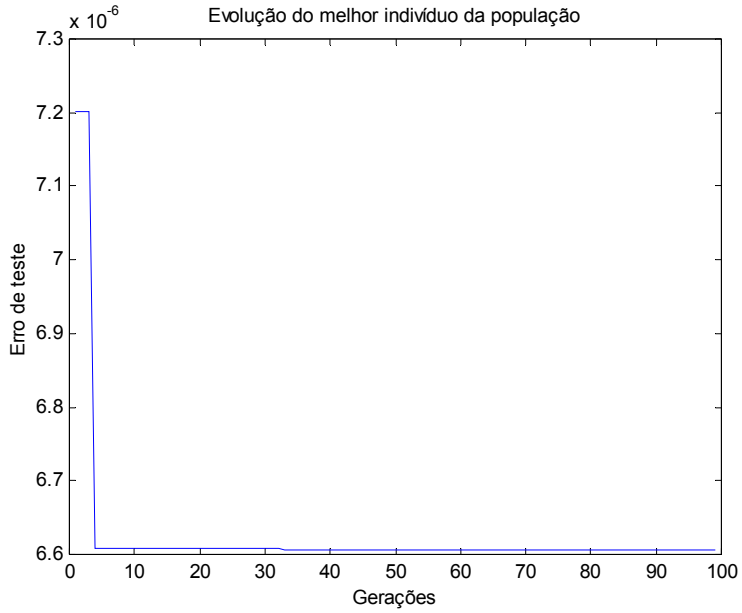


Figura 6.17: Representação gráfica da evolução da população para o modelo directo treinado com *Early Stopping*.

de *Early Stopping* e as figuras 6.19 e 6.20 dizem respeito à utilização da técnica de Regularização.

Como é possível verificar pelas figuras 6.17, 6.18, 6.19 e 6.20 e evolução da população é bastante rápida (leva cerca de 100 gerações) e pelos resultados obtidos em termos de erro de teste os modelos prometem ser de elevada qualidade.

Para obter os parâmetros dos modelos foi necessário proceder à implementação das RNs. No anexo B está exemplificada a forma como é feita essa implementação, primeiro com uma função que calcula a saída do modelo e depois com uma função que simula Controlo com Modelo Inverso.

6.13 Conclusões

Neste capítulo foram analisadas as características do sistema utilizado como caso prático, descrito a electrónica que o compõe, apresentado o código base de acesso a esse mesmo *hardware* e descritas as opções base dos modelos neuronais criados.

Foi também apresentado um procedimento automático baseado em AGs que permite a optimização de modelos de forma automatizada que foi utilizado para melhorar a qualidade dos modelos, escolher a melhor estrutura e calcular os parâmetros necessários para as técnicas de regularização implícita e explícita.

Como é possível verificar pelos resultados apresentados, todos os modelos neuronais apresentados obtiveram valores de EQM muito mais baixos do que os conseguidos com os modelos lineares. Acresce ainda que os modelos optimizados com AGs são muito

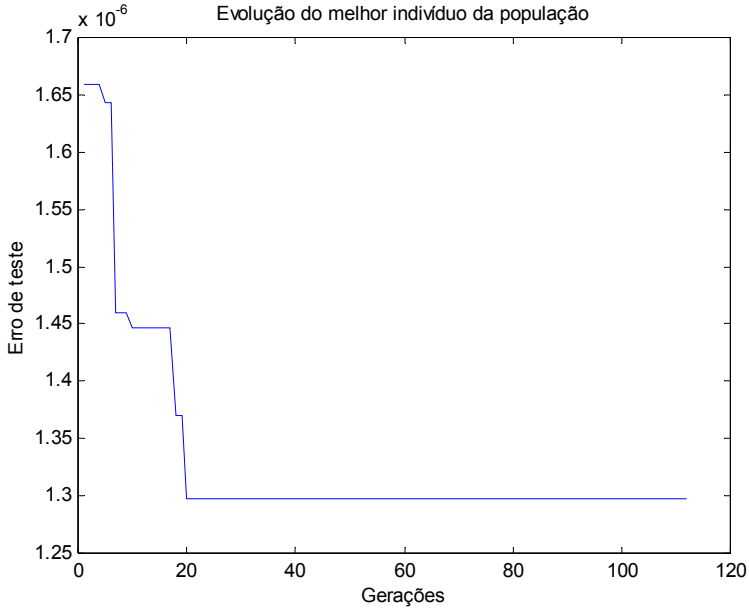


Figura 6.18: Representação gráfica da evolução da população para o modelo inverso treinado com *Early Stopping*.

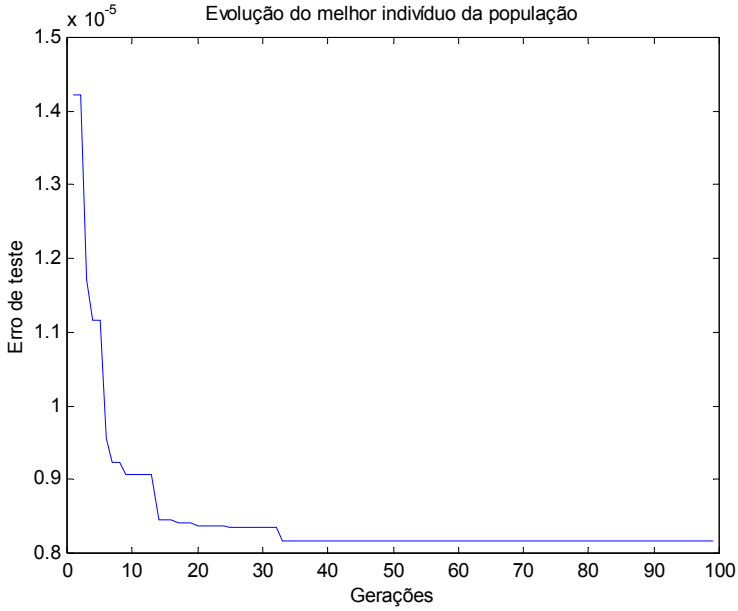


Figura 6.19: Representação gráfica da evolução da população para o modelo directo treinado com Regularização.

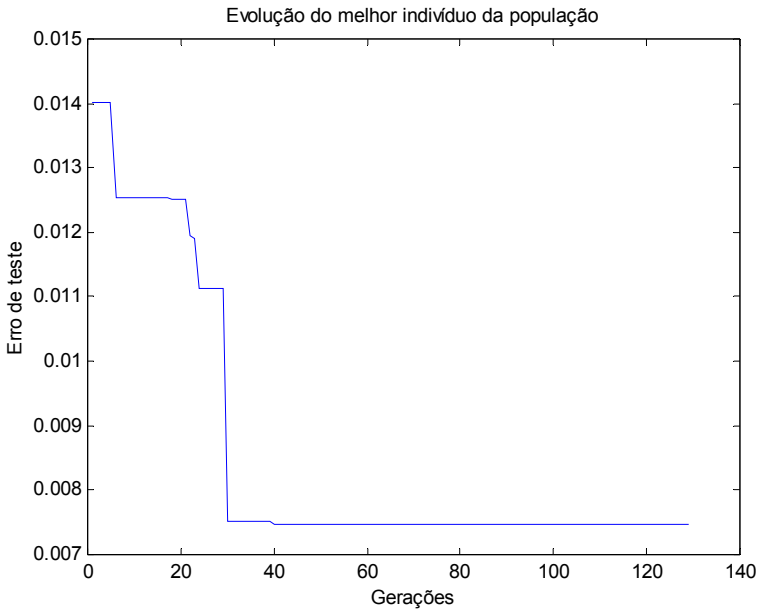


Figura 6.20: Representação gráfica da evolução da população para o modelo inverso treinado com Regularização.

semelhantes e consideravelmente melhores do que o modelo optimizado pelo operador humano.

Capítulo 7

Resultados de controlo

"Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information upon it." - Samuel Johnson, em Boswell's Life of Johnson, escritor. (1709 - 1784)

7.1 Introdução

Neste capítulo são apresentados os resultados de controlo, obtidos com os modelos referidos no capítulo 6, em diversas malhas de controlo e apresentados também resultados de identificação e controlo *on-line*. A primeira parte do capítulo é composta de resultados de controlo com modelos treinados *off-line* e a segunda por resultados de controlo com identificação *on-line*.

São também apresentados os resultados de algumas das contribuições introduzidas nesta tese, nomeadamente no que diz respeito à malha AIMC e à identificação *on-line* com o algoritmo de Levenberg-Marquardt, em janela deslizante com *Early Stopping*, na secção intitulada Identificação *on-line*. Parte do trabalho descrito nessa secção foi apresentado no artigo "Implementando o Algoritmo de Levenberg-Marquardt *On-line*: uma Solução em Janela Deslizante com *Early Stopping*" (o título original em inglês é "*Implementing the Levenberg-Marquardt Algorithm on-line: a Sliding Window Approach with Early Stopping"),* publicado no 2^{nd} *IFAC Workshop on Advanced Fuzzy/Neural Control* [96].

Os resultados de identificação e controlo *on-line* são também uma boa demonstração da utilidade da malha de AIMC genérica, uma vez que mostram a versatilidade desta estrutura para permitir a comutação entre controladores.

7.2 Modelos optimizados pelo operador humano

Estes primeiros modelos obtidos foram optimizados utilizando os conhecimentos do operador humano relativamente às técnicas de modelização e às RNs, serão designados por modelos optimizados pelo operador humano, e foram testados em diversas malhas de controlo, cujos resultados são apresentados a seguir.

Na figura 7.1 são apresentados os resultados de Controlo com Modelo Inverso (DIC) com os modelos optimizados pelo operador humano. Nesta experiência e nas seguintes, utilizou-se a mesma referência (ou *set point*) para todas as malhas de controlo para permitir a comparação dos diversos resultados de forma directa.

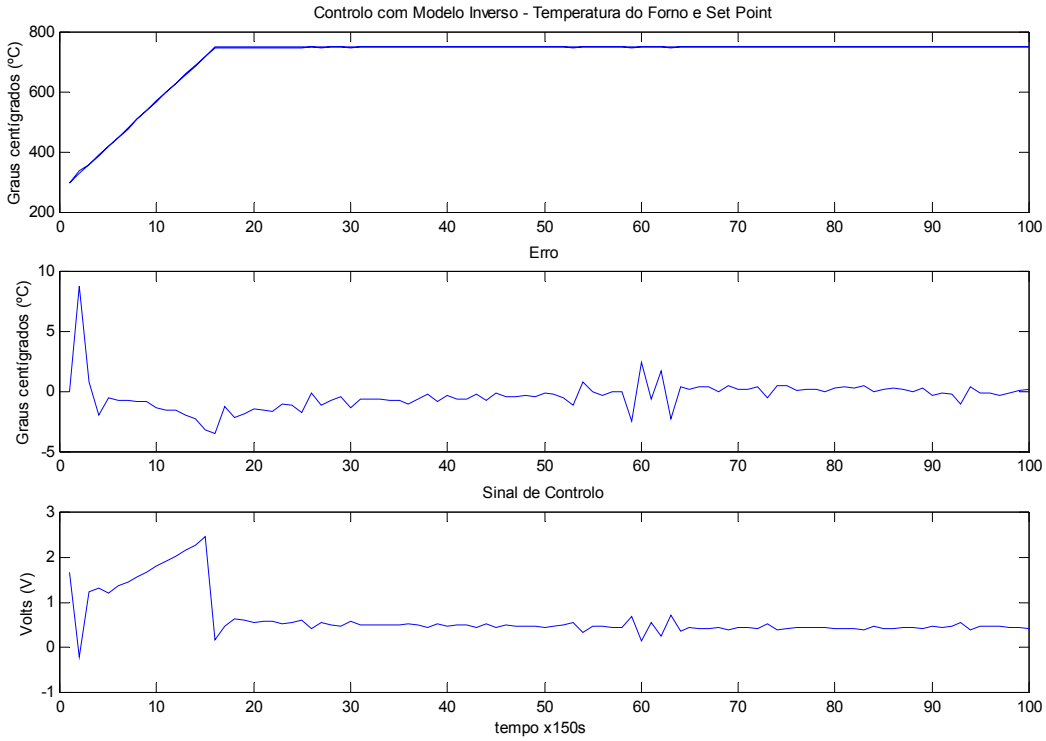


Figura 7.1: Resultado de Controlo com Modelo Inverso (DIC) utilizando modelos optimizados pelo operador humano.

Na figura 7.2 é possível ver os resultados de controlo utilizando uma malha de Controlo Baseado em Modelo Interno (IMC).

Na figura 7.3 é apresentado um resultado de controlo com a malha do Controlador Aditivo misto (AFC), (ver secção 5.4.2) onde é utilizado como controlador existente um controlador do tipo *Proportional Integral* - PI, com os seguintes parâmetros: Kp=0,01 e Ki=0,01.

Existem, e estão descritos na literatura, diversos métodos de sintonia de controladores PID, que vão desde a sintonia manual feita pelo operador humano ao conhecido método de Ziegler-Nichols e, mais recentemente, os métodos de sintonia automática, por exemplo com aproximações de *soft computing* como sejam os AGs e as RNs (ver, por exemplo, [97] e [98]). Ao longo deste trabalho este assunto também foi abordado, embora de forma breve, como pode ser visto em [99]. Como é verificado nesse trabalho a optimização de um controlador deste tipo não é genérica, mas sim dependente do sinal de referência em questão na experiência, principalmente no caso de um sistema

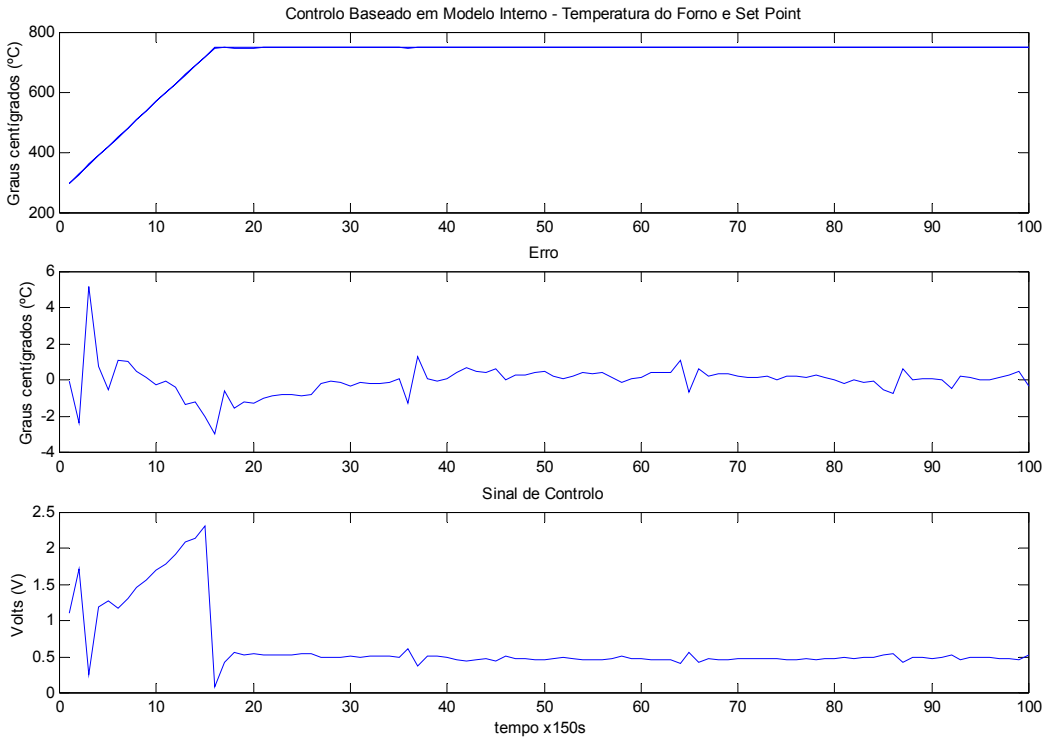


Figura 7.2: Resultado de Controlo Baseado em Modelo Interno (IMC) utilizando modelos optimizados pelo operador humano.

não linear. Nesta situação existe uma vantagem clara de soluções com RNs uma vez que estas têm capacidade de generalizar.

O controlador PI utilizado ao longo deste capítulo foi optimizado pelo operador humano.

Nesta experiência o controlador PI é desligado após a amostra 200. O objectivo desta alteração é verificar não só o funcionamento da malha de AFC, como observar a transição para o funcionamento em DIC e verificar se essa transição introduz algum período de oscilação. Como é possível verificar pela imagem 7.3, o funcionamento do AFC é "bastante bom" e não ocorrem quaisquer problemas na transição.

Na figura 7.4 é apresentado um resultado de controlo com a malha de Controlo Aditivo Baseado em Modelo Interno (AIMC), onde é utilizado como controlador existente o mesmo PI do exemplo anterior. Nesta experiência o controlador PI é também desligado após a amostra 200, com o mesmo objectivo do caso anterior, sendo nesta situação a transição para IMC.

Como é possível verificar pela figura 7.4, a malha de Controlo Aditivo Baseado em Modelo Interno funciona bem tanto em conjunto com o PI, como após a sua remoção (a partir da amostra 200). A transição, não dá origem a oscilação nem a qualquer outro problema.

Para que possa ser analisado o controlador existente utilizado, apresenta-se na figura

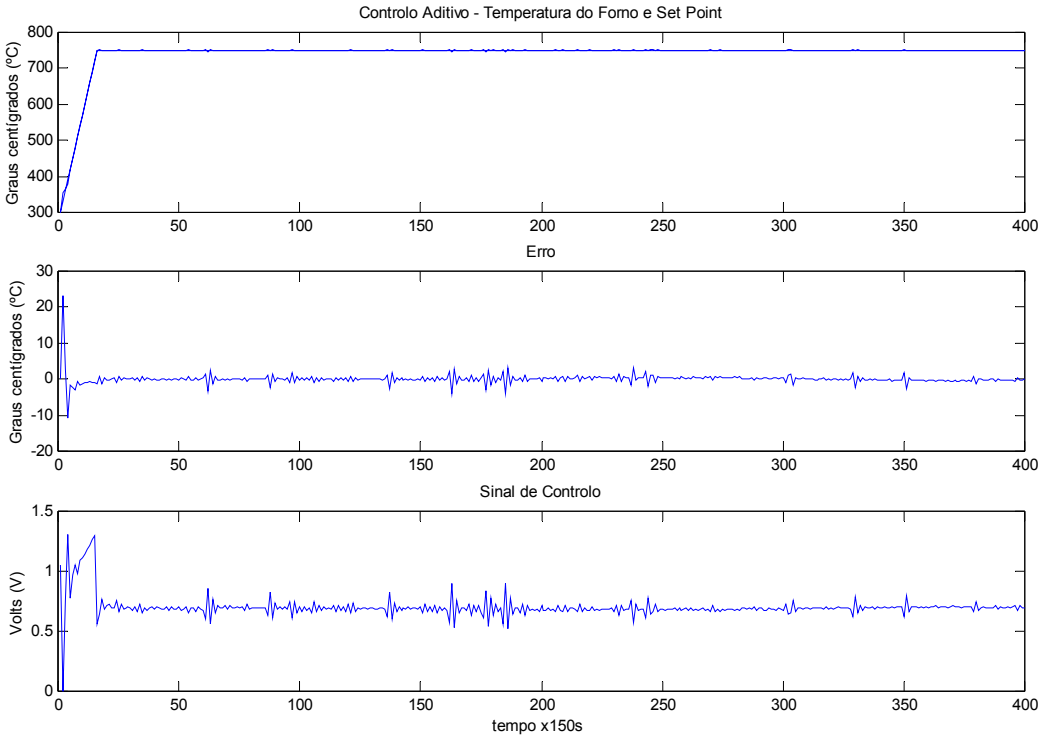


Figura 7.3: Resultado de Controlo Aditivo (AFC).

7.5, o resultado de uma experiência em que foi aplicado o controlador PI isoladamente.

Como é possível verificar, com o controlador PI, para este sinal de referência, é obtido controlo de qualidade "aceitável", ainda que a oscilação inicial seja maior do que nos restantes casos.

Na tabela 7.1 é apresentado um resumo dos valores de Erro Quadrático Médio (EQM) para as diversas soluções de controlo testadas com os modelos optimizados pelo operador humano. Estes valores foram divididos por forma a apresentar, inicialmente, o erro no total da experiência e depois o erro após a fase inicial de subida da temperatura. Esta opção foi tomada devido a um certo carácter aleatório desta fase inicial de aquecimento, ou seja, em algumas experiências, nesta fase, existe um desvio em relação ao sinal de referência, nem sempre com uma justificação compreensível, que mascara a restante parte dos resultados.

Pelos resultados apresentados pode verificar-se a elevada qualidade obtida com o controlo baseado nas RNs, uma vez que todos os resultados apresentam um erro inferior a 1ºC, apesar da existência de ruído provocar uma certa incerteza que deve ser reflectida na análise dos resultados.

Como seria expectável, o Controlo Baseado em Modelo Interno apresenta melhores resultados do que o Controlo com Modelo Inverso e correspondentemente o Controlo Aditivo Baseado em Modelo Interno apresenta também melhores resultados do que o Controlo Aditivo.

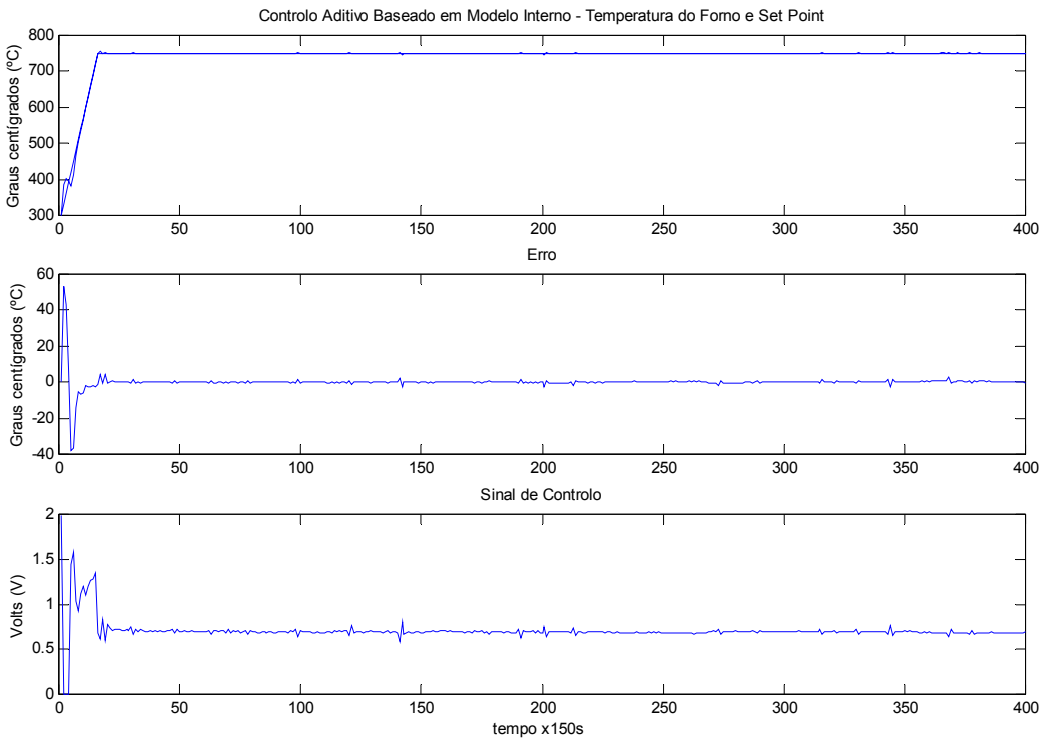


Figura 7.4: Resultado de Controlo Aditivo Baseado em Modelo Interno utilizando modelos optimizados pelo operador humano (AIMC).

Estratégia\EQM	Amostras 1 a 100	Amostras 16 a 100
DIC	1,81	0,84
IMC	0,75	0,36
AFC	7,29	0,53
AIMC	25,93	0,46
PI	96,80	29,73

Tabela 7.1: Sumário dos valores de EQM para modelos optimizados pelo operador humano.

No Controlador Aditivo e no Controlador Aditivo Baseado em Modelo Interno é especialmente notório que a fase de arranque inicial contribui com um erro muito elevado, o que é compreensível sabendo que é a fase de arranque de dois controladores em simultâneo que calculam o valor do sinal de controlo, para a primeira amostra, sem a percepção de que existe um outro controlador ligado ao sistema.

Como é possível verificar pelos resultados apresentados, o controlo com o PI é o que apresenta o valor de erro mais elevado, apesar de ter sido optimizado pelo operador humano com vista a esta referência.

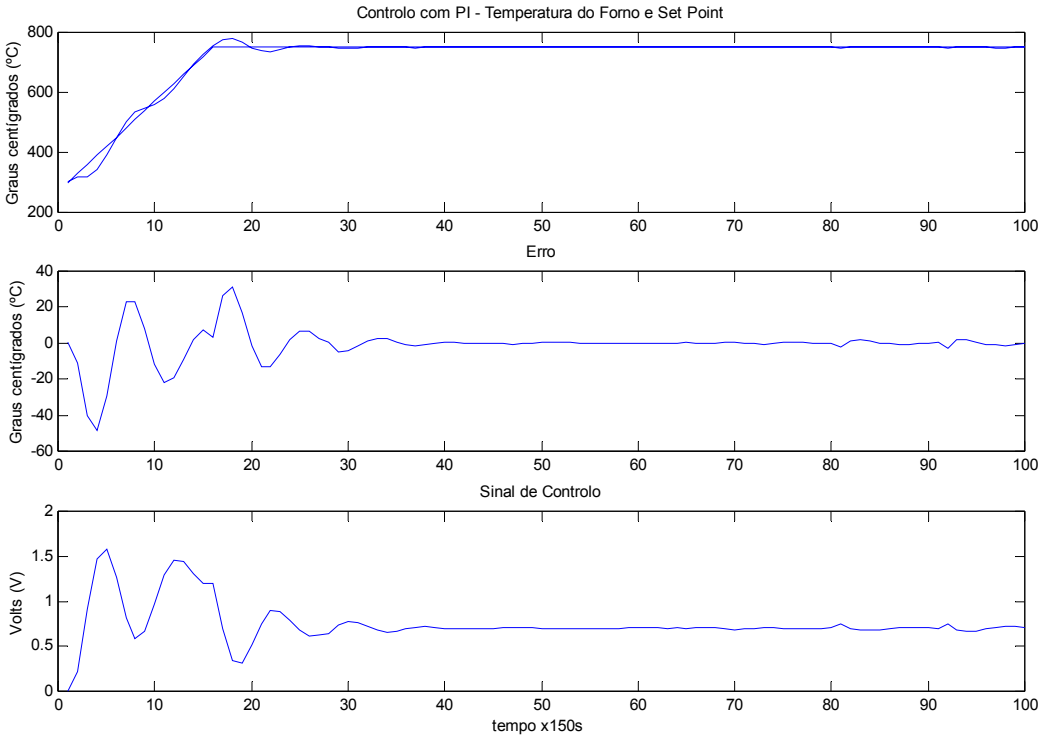


Figura 7.5: Controlo do sistema de teste por intermédio do controlador PI.

7.3 Modelos optimizados com Paragem de Treino Antecipada

Nesta secção são apresentados os resultados de controlo obtidos com os modelos optimizados com Paragem de Treino Antecipada. Nesta secção e na secção seguinte, são apresentados apenas os resultados de controlo obtidos com as malhas de controlo DIC e IMC. Esta redução foi efectuada por uma questão de espaço e a opção por estas duas malhas foi feita por serem as mais divulgadas na literatura.

Na figura 7.6 é possível observar um resultado obtido com uma malha de controlo com modelo inverso.

Na figura 7.7 pode ver-se o resultado da aplicação da malha IMC para controlo do sistema.

A tabela 7.2 apresenta valores do EQM para os modelos com Paragem de Treino Antecipada ou *Early Stopping*, comparados com os valores obtidos anteriormente para os modelos optimizados pelo operador humano.

Como é possível verificar, no que diz respeito ao EQM, existe uma melhoria considerável (cerca de 20%) da optimização feita pelo operador humano para a optimização feita com AGs para a técnica de *Early Stopping*. Estes valores são mais fáceis de comparar de forma directa uma vez que o operador humano utiliza também *Early*

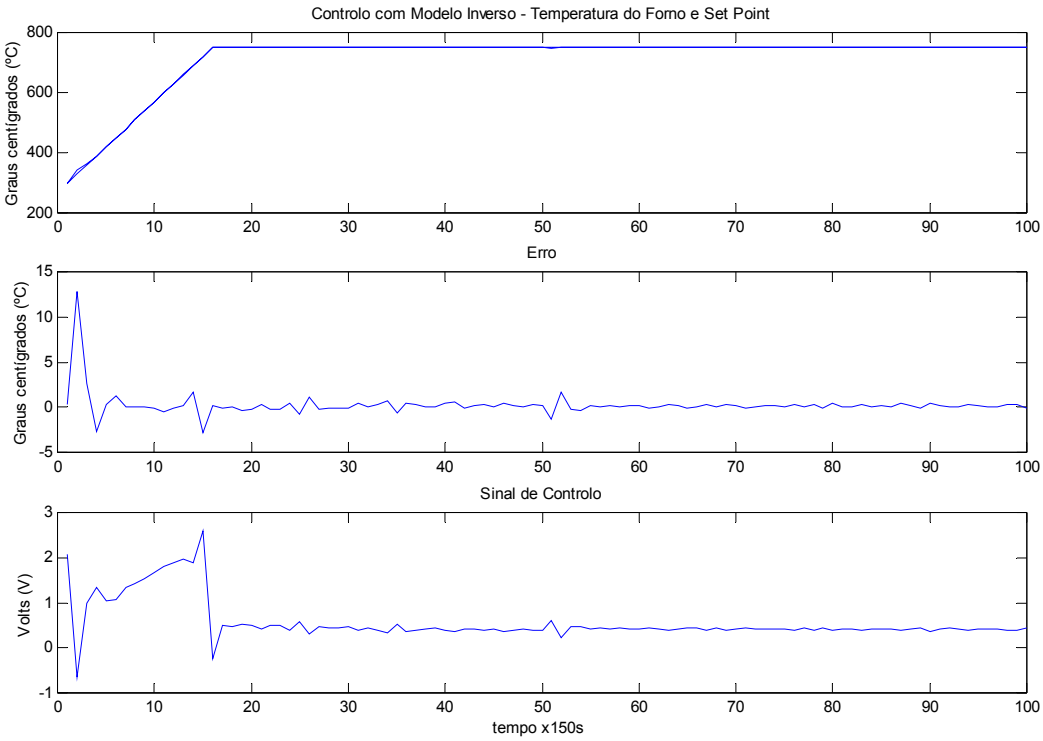


Figura 7.6: Resultado de Controlo com Modelo Inverso (DIC) utilizando modelos optimizados com Paragem de Treino Antecipada.

Estratégia\EQM	Amostras 1 a 100	Amostras 16 a 100
DIC	1,81	0,84
IMC	0,75	0,36
DIC (ES)	2,89	0,71
IMC (ES)	2,37	0,30

Tabela 7.2: Sumário dos valores de EQM para modelos optimizados com Paragem de Treino Antecipada.

Stopping.

7.4 Modelos optimizados com Regularização

Os modelos optimizados com Regularização apresentados no capítulo anterior foram utilizados também em malhas de controlo. Na figura 7.8 é possível ver um resultado de controlo com modelo inverso utilizando modelos treinados com regularização.

A figura 7.9 apresenta os dados obtidos numa experiência de Controlo Baseado em Modelo Interno, com os modelos optimizados com Regularização.

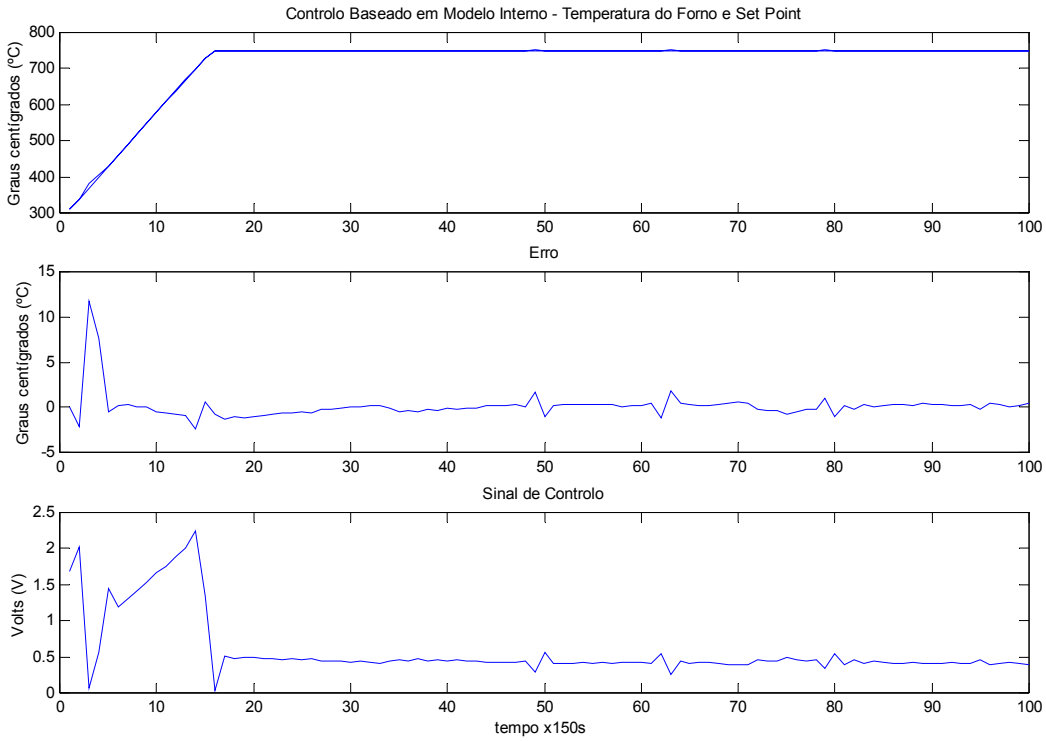


Figura 7.7: Resultado de Controlo Baseado em Modelo Interno (IMC) utilizando modelos optimizados com Paragem de Treino Antecipada.

Estratégia\EQM	Amostras 1 a 100	Amostras 16 a 100
DIC	1,81	0,84
IMC	0,75	0,36
DIC (Regul.)	1,11	0,49
IMC (Regul.)	11,69	0,38

Tabela 7.3: Sumário dos valores de EQM para modelos optimizados com Regularização.

A tabela 7.2 apresenta valores do EQM para os modelos optimizados com Regularização, comparados com os valores obtidos anteriormente para os modelos optimizados pelo operador humano.

Para a técnica de Regularização é possível verificar que se obtêm também melhores resultados, principalmente no que diz respeito ao DIC, já que no caso do IMC o resultado é praticamente igual ao obtido com os modelos iniciais. É difícil nesta situação estabelecer uma conclusão clara sobre qual das técnicas permite obter melhores resultados. Tendo sido utilizadas duas malhas de controlo, é obtido o melhor resultado em cada uma das malhas com uma técnica diferente. Se estiver disponível uma solução que permita a obtenção dos parâmetros necessários a ambas as técnicas será recomendável testar ambas as soluções. Se tal não acontecer é preferível optar pela técnica de Para-

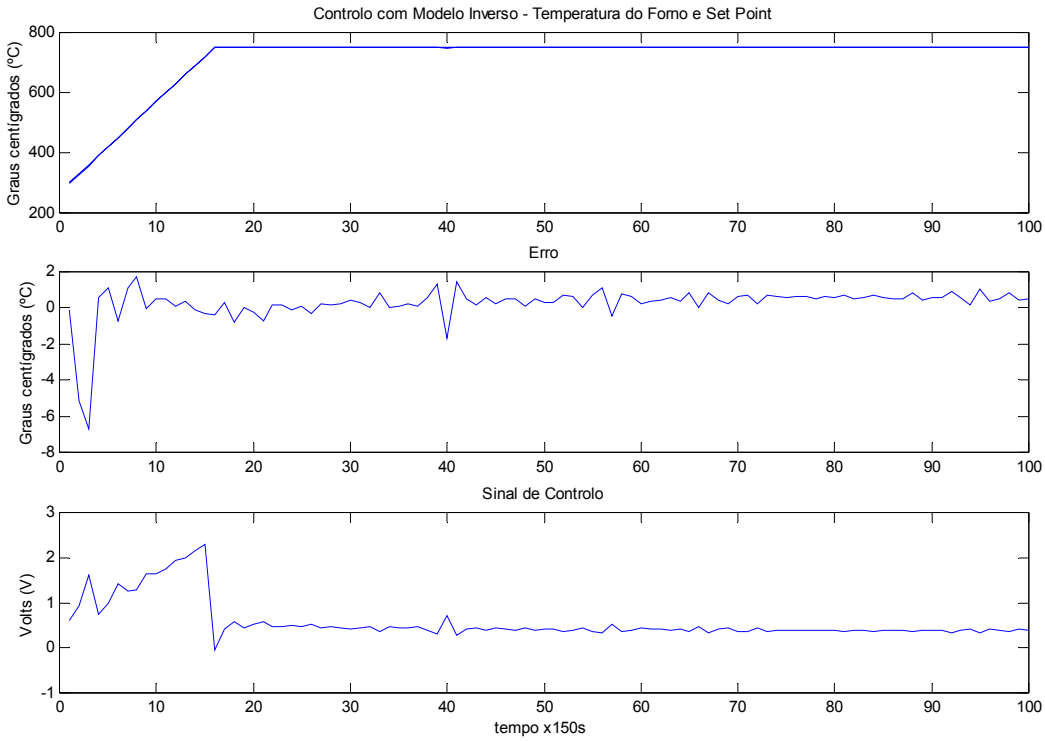


Figura 7.8: Resultado de Controlo com Modelo Inverso (DIC) utilizando modelos optimizados com Regularização.

gem de Treino Antecipada uma vez que é mais fácil obter uma solução razoável com esta alternativa.

7.5 Controlo com identificação *on-line*

7.5.1 Introdução

Como foi abordado no capítulo 2, é possível implementar qualquer algoritmo em versão *on-line* e *off-line*. Para certos algoritmos existem dificuldades adicionais na implementação *on-line*. A utilidade da identificação de sistemas *on-line* prende-se principalmente com o controlo de sistemas variantes no tempo e com o desenvolvimento de aproximações do tipo caixa preta, onde não é necessária nenhuma informação prévia sobre o sistema a modelizar.

Entre os diversos algoritmos utilizados no treino de RNs, o algoritmo de Levenberg-Marquardt (ver capítulo 2) tem sido considerado como o mais eficaz, mas o seu uso tem sido restringido à utilização *off-line* devido às dificuldades em implementar uma versão iterativa que mantenha as qualidades deste algoritmo. Estas dificuldades resultam do cálculo das derivadas para a matriz Hessiana, da necessidade de inverter essa matriz e

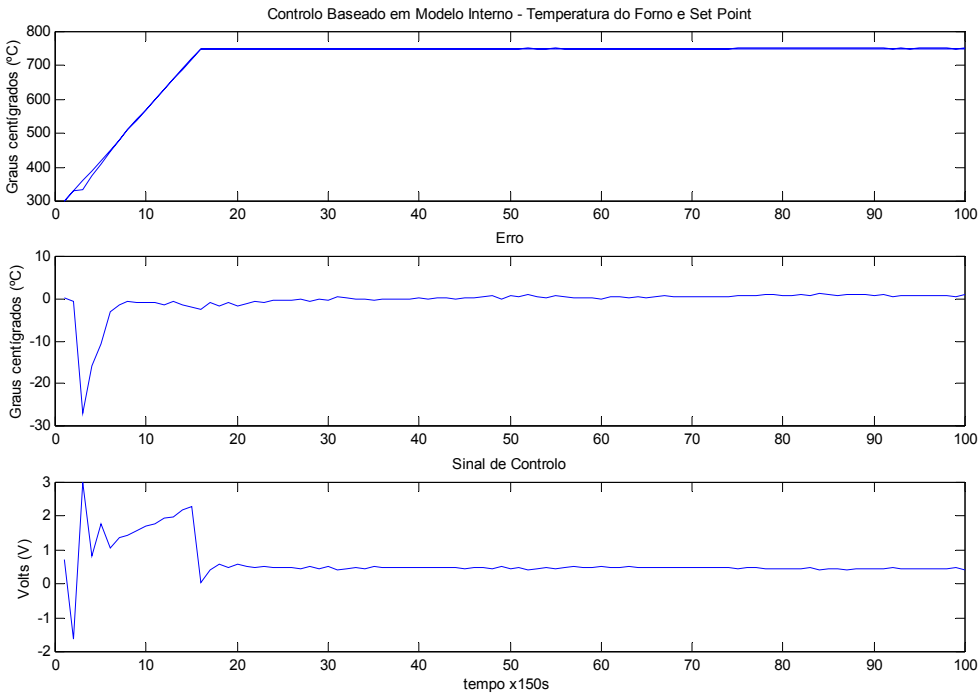


Figura 7.9: Resultado de Controlo Baseado em Modelo Interno (IMC) utilizando modelos optimizados com Regularização.

do cálculo da região de validade da aproximação contida no cálculo da matriz Hessiana (designada habitualmente por região de confiança).

No caso deste trabalho é proposta uma abordagem diferente: a utilização do algoritmo de Levenberg-Marquardt *on-line* através de uma versão em grupo com uma janela deslizante e paragem de treino antecipada para evitar o treino excessivo. Esta implementação também utiliza a técnica híbrida genérica/especializada para modelos inversos.

Esta solução foi utilizada em duas versões: com janela de teste fixa ou deslizante.

7.5.2 Outras implementações *on-line*

Na literatura existem outras tentativas de construir implementações *on-line* para o algoritmo de Levenberg-Marquardt. Entre esses trabalhos vale a pena realçar o trabalho feito por Ngia ([33] e [34]), desenvolvendo uma versão iterativa modificada do algoritmo de Levenberg-Marquardt, que inclui o cálculo da região de confiança e o trabalho apresentado em [32], que implementa uma versão do algoritmo, em janela deslizante, utilizando RBFs.

Janela deslizante e Paragem de Treino Antecipada

Os modelos de identificação *on-line*, foram obtidos utilizando o algoritmo de Levenberg-Marquardt em janela deslizante com Paragem de Treino Antecipada, com duas variantes no que diz respeito à janela de teste. Existe uma versão em que a janela de teste é fixa, composta por um conjunto de pontos considerado representativo da zona de funcionamento do sistema que é previamente preparado e uma versão em que a janela de teste, tal como a janela de treino, é também deslizante.

A técnica de Paragem de Treino Antecipada é usada aqui para evitar o problema do treino excessivo e foi escolhida entre as possibilidades descritas no capítulo 3 por obrigar a um menor esforço computacional. Uma solução para evitar o treino excessivo é indispensável quando se está a trabalhar com sistemas sujeitos a ruído.

A utilização da Paragem de Treino Antecipada não é possível de forma tão directa como acontece quando os modelos são treinados *off-line*, uma vez que o conjunto de treino está em mudança ao longo das iterações, tal como está exemplificado na figura 7.10, onde as duas situações possíveis estão representadas, com a variável m_{epoca} a ser utilizada como índice das iterações.

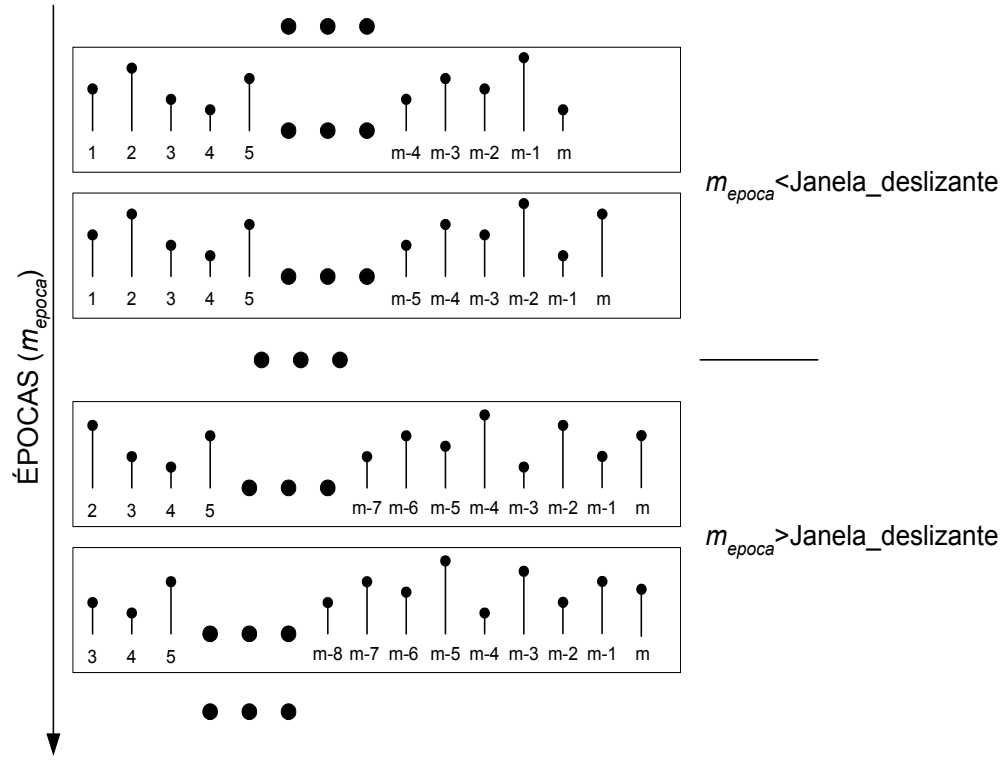


Figura 7.10: Conteúdo da janela deslizante em função da evolução das épocas.

O treino é iniciado após a recolha de alguns dados, mas antes de ser recolhida a quantidade de dados necessária para completar a janela deslizante, para diminuir o tempo inicial de espera. Nesta fase inicial a janela deslizante está a crescer, embora o

ritmo de crescimento não corresponda a uma amostra por cada época de treino, uma vez que todo o tempo de amostragem é utilizado para o treino, sendo efectuadas diversas épocas antes de estar disponível uma nova amostra para integrar a janela deslizante.

Na segunda fase, após o número de amostras recolhido exceder o tamanho da janela deslizante, a janela deslizante muda devido à entrada de uma nova amostra e à remoção de uma amostra por cada período de amostragem.

Estas mudanças frequentes no conteúdo da janela deslizante implicam que, em algumas situações, o valor do erro de teste possa sofrer "saltos" de uma iteração para a seguinte, o que pode ser interpretado erradamente como uma situação de treino excessivo, obrigando a um cuidado redobrado na detecção deste tipo de situação.

O procedimento utilizado para a identificação *on-line* do modelo directo está representada na figura 7.11.

Como foi explicado, o treino tem início quando estão disponíveis amostras em número correspondente ao valor pré-definido, sendo este inferior ao tamanho da janela deslizante. Após cada época, a RN é avaliada através de uma sequência de teste. O valor do EQM obtido é utilizado para a técnica de Paragem de Treino Antecipada e para reter o melhor modelo. O treino é iniciado com pesos aleatórios com valores entre 0 e 1. Estes valores foram escolhidos porque as RNs vão trabalhar com entradas escaladas.

As condições de treino excessivo e de número máximo de épocas são então verificadas e, caso alguma delas seja verdadeira, é verificado o indicador que informa se o limiar de qualidade foi atingido. Caso esta condição seja verdadeira é iniciado o treino do modelo inverso e, no caso de ser falsa, o modelo é apagado para que sejam preparados novos modelos. Apagar o modelo significa aqui recomeçar o treino com pesos aleatórios nas mesmas condições dos pesos iniciais.

Após o teste das condições de treino excessivo e de número máximo de épocas, no caso de serem ambas falsas, é testada uma condição pré-definida relativa ao erro máximo aceitável e, no caso de esse limiar já ter sido atingido, um indicador apropriado é activado. Em qualquer dos casos é avaliado o tempo restante do período de amostragem para decidir se é possível treinar mais uma época antes do instante de amostragem ou se é necessário proceder à amostragem e inclusão da nova amostra na janela deslizante para que o treino da nova época já seja feito com a nova actualização.

O limiar pré-definido que é utilizado é uma grandeza física que foi definida na fase de projecto: trata-se do valor do erro máximo em graus centígrados que é aceitável.

Os procedimentos são muito semelhantes tanto no casos dos modelos inversos, como no caso das duas variantes relativas à janela deslizante (no caso da figura 7.11 trata-se de uma situação de janela de teste fixa), estando implementada a comutação consecutiva entre modelo directo e modelo inverso. As diferenças residem apenas na forma de avaliação dos modelos inversos, que é feita de acordo com a secção 3.6.2 (enquanto os modelos directos são avaliados utilizando a sequência de teste), e na utilização ou não da sequência de teste em janela deslizante.

Quando são utilizadas duas janelas deslizantes, uma para o treino e outra para o teste, é necessário escolher a posição relativa dessas janelas. Para que seja possível efectuar Paragem de Treino Antecipada de forma válida entendeu-se que a solução

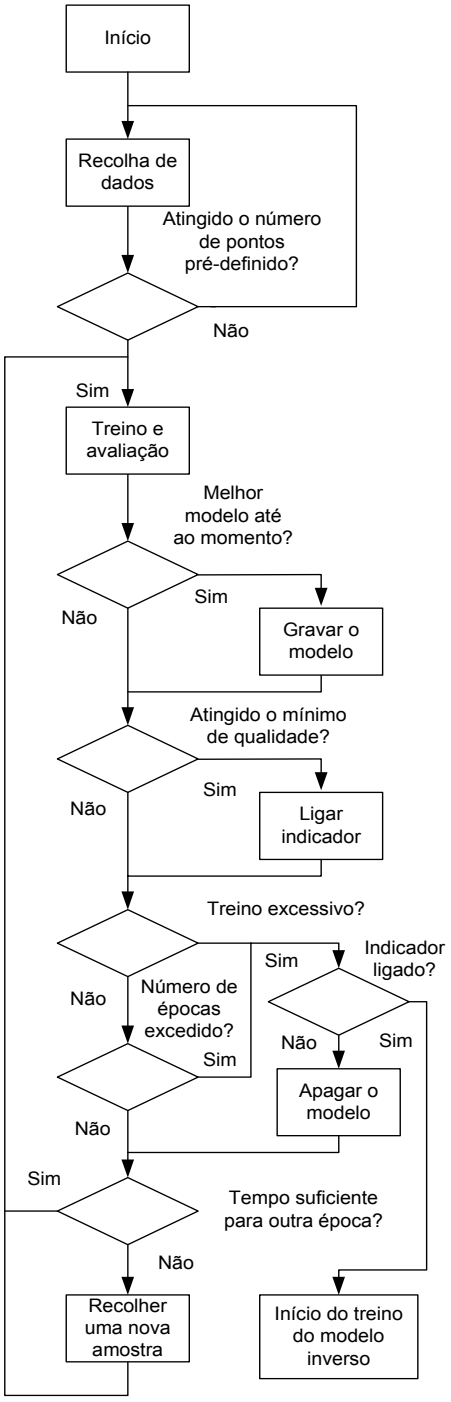


Figura 7.11: Diagrama de blocos representativo do algoritmo de identificação on-line.

correcta seria colocar as novas amostras na janela de teste e as amostras que fossem retiradas da janela de teste entrariam na janela de treino, de acordo com o que está documentado na figura 7.12.

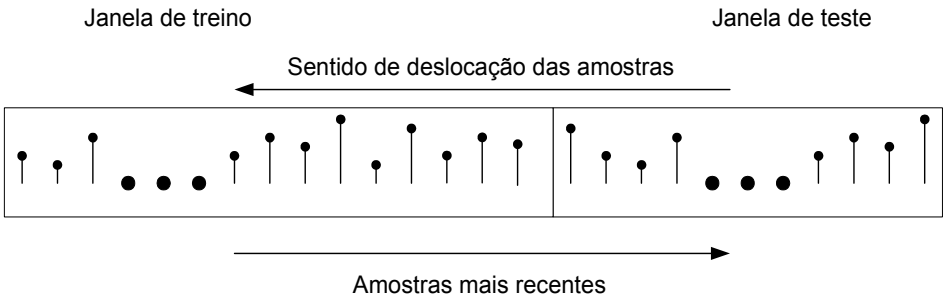


Figura 7.12: Disposição das janelas de treino e de teste, com indicação do sentido de deslocamento das amostras.

Caso fosse utilizada a disposição inversa das janelas, as amostras, depois de serem utilizadas no treino passariam a fazer parte da sequência de teste, pelo que o objectivo de avaliar a capacidade de generalização da RN seria falseado.

A primeira versão do algoritmo, com a janela de teste fixa, foi encarada como um passo intermédio de passagem do treino *off-line* para o treino *on-line*. A manutenção da janela de teste fixa permitia que fossem analisados inicialmente apenas os problemas decorrentes de ter um conjunto de treino que não está fixo, sendo analisados os problemas decorrentes de ter um conjunto de teste também variável numa fase posterior.

A janela de teste fixa não permite a identificação de sistemas variantes no tempo, uma vez que, embora o treino pudesse incluir as novas amostras representativas das variações ocorridas no sistema, a janela de teste estaria "presa" às características iniciais.

Nas figuras 7.13, 7.14 e 7.15 é possível ver os resultados de identificação e controlo com as estratégias de Controlo com Modelo Inverso, Controlo Baseado em Modelo Interno e Controlo Aditivo Baseado em Modelo Interno, respectivamente, para a primeira versão do algoritmo, ou seja, com a janela de teste fixa.

Os modelos utilizados têm as seguintes características: dois regressores do sinal de entrada e dois regressores do sinal de saída, seis neurónios na camada escondida, uma saída linear e valor inicial do parâmetro $\mu = 1$ (ver secção 2.7.1). O treino foi iniciado após a recolha de 140 amostras, sendo a janela de treino composta por um máximo de 250 amostras (inicialmente estão apenas disponíveis as 140 amostras referidas) e utilizado um conjunto previamente preparado de 150 amostras como sequência de teste fixa.

Antes da identificação estar completa e ser possível o controlo com os modelos neuronais o funcionamento do sistema é assegurado pelo controlador PI já anteriormente descrito. O principal objectivo da utilização do controlador PI é colocar o sistema numa zona de funcionamento apropriada para que a identificação se possa fazer de

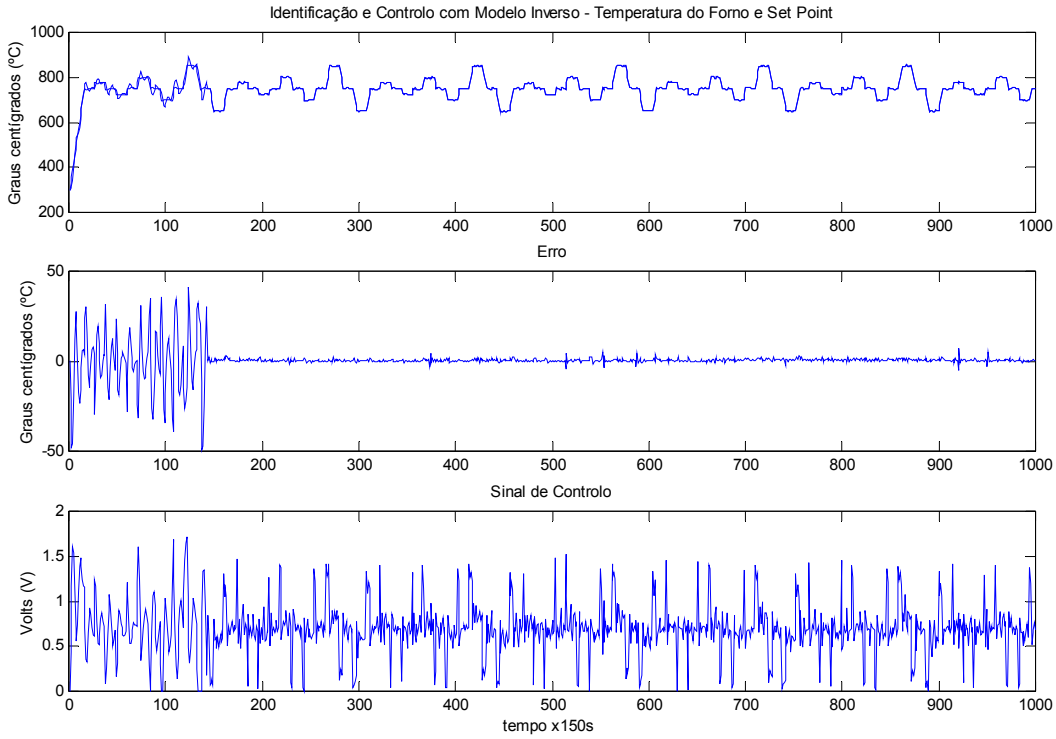


Figura 7.13: Resultado de identificação *on-line* e Controlo com Modelo Inverso (DIC). A fase inicial de controlo é assegurada por um PI.

forma correcta.

A utilização de um controlador do tipo PI tem ainda uma outra utilidade: como foi analisado na secção 3.3.1 existem algumas dificuldades em proceder à aplicação de um factor de escala sobre um sinal cuja forma exacta, média e variância são desconhecidos. A existência do controlador PI pode facilitar a resolução deste problema, uma vez que a fase inicial da experiência pode fornecer valores aproximados da média e da variância, ou seja, o controlador PI, que não é afectado por nenhum factor de escala, pode funcionar na fase inicial sem nenhum problema, sendo usado o controlador neuronal quando já existem dados que permitem calcular um valor de média e variância. Deve notar-se que ao fazer a alteração do controlador tipo PI para qualquer uma das outras malhas, se está a fazer uso da malha AIMC genérica. Relativamente à figura 5.14, com o controlador PI está apenas fechado o interruptor 1, sendo PI o controlador existente. Para passar ao Controlo com o Modelo Inverso é fechado o interruptor 2 e aberto o 1.

O sinal de referência ou *set point* escolhido pretende simular um perfil de temperatura utilizado para a cozedura de algumas peças da indústria cerâmica, tendo sido adicionado a este sinal um pequeno sinal de *dithering* para manter a excitação no sinal de controlo e facilitar o treino da RN que, caso contrário, poderia não dispor de informação suficiente para treinar modelos de boa qualidade. Como foi referido na secção 3.2.3, para sistemas não-lineares, é necessário escolher não só um espectro de frequência

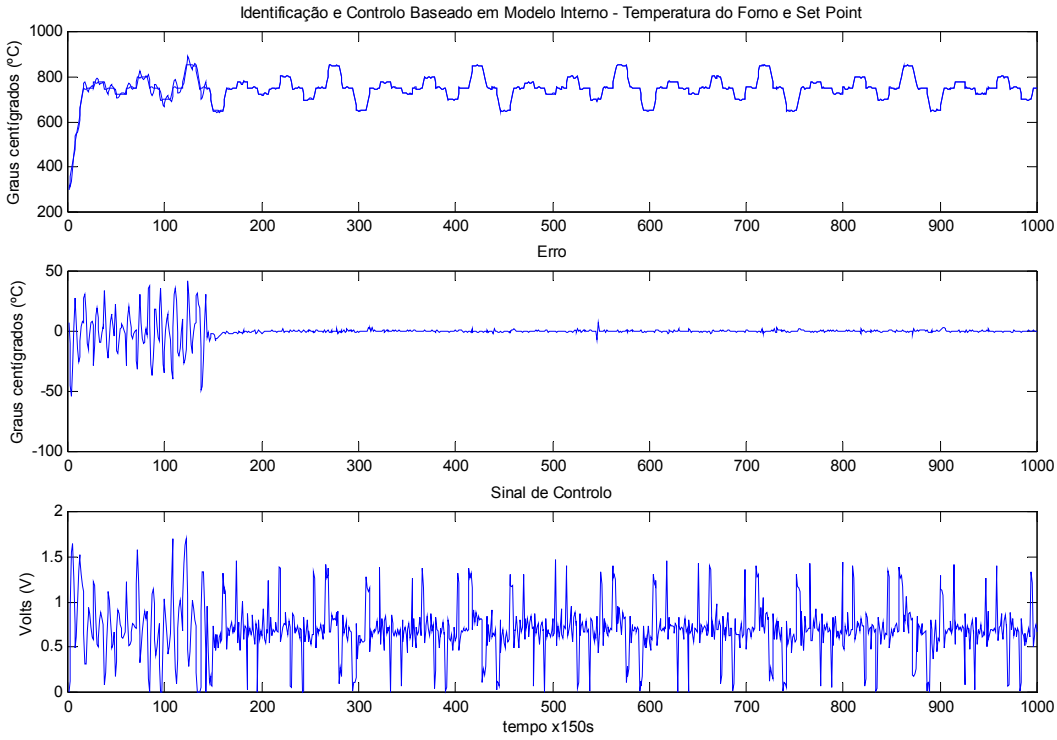


Figura 7.14: Resultado de identificação *on-line* e Controlo Baseado em Modelo Interno (IMC). A fase inicial de controlo é assegurada por um PI.

alargado como também uma gama de amplitudes abrangente, que não é possível obter a partir de um sinal de referência que possui apenas sete valores distintos.

Este sinal foi mantido para todas as experiências de treino *on-line* que são apresentadas ao longo deste capítulo para simplificar a análise visual dos resultados.

Como é possível verificar pelos resultados apresentados, os modelos que foram obtidos permitem a implementação das malhas de controlo com qualidade muito superior ao controlo efectuado pelo PI na fase inicial, cujo sinal é utilizado para a identificação.

Para facilitar a análise da implementação do algoritmo de Levenberg-Marquardt com Paragem de Treino Antecipada, em janela deslizante, com a janela de teste fixa, apresenta-se, de forma muito simplificada, o algoritmo para o caso de Controlo com Modelo Inverso.

```
Carregar o sinal de referência
Inicializar a comunicação via porta série
Fazer o aquecimento inicial do forno (até aos 300ºC)
Configurar os modelos iniciais (Directo e Inverso)
Gerar pesos aleatórios
Desde i=1 até número_de_iterações
    Fazer leitura de temperatura
```

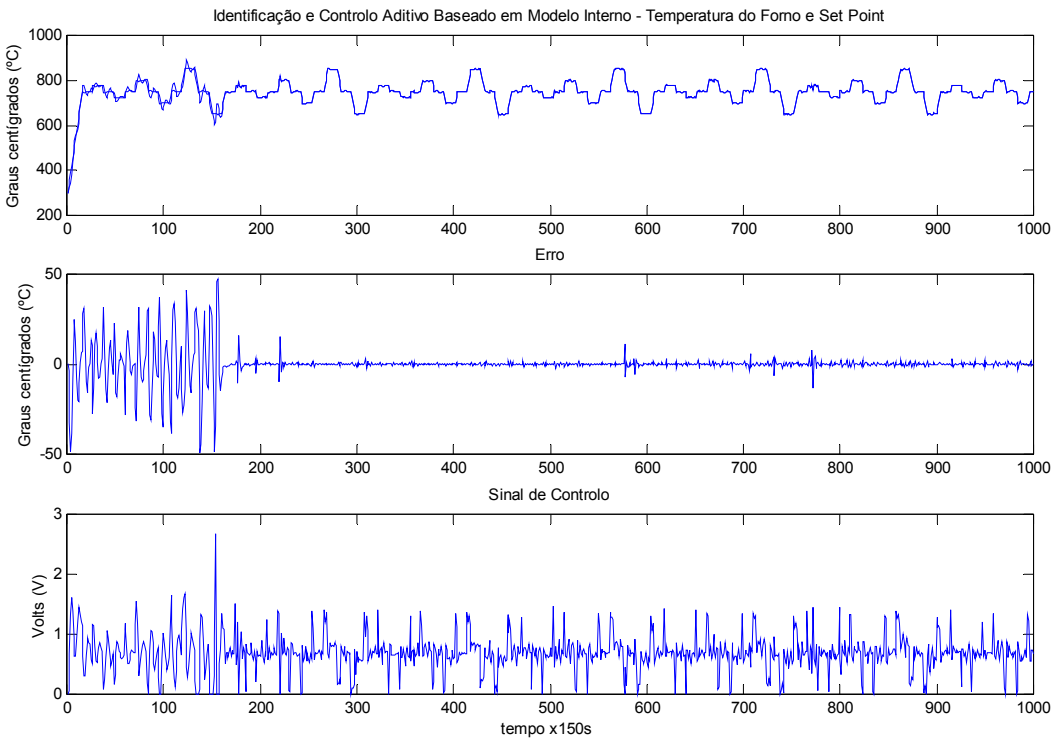


Figura 7.15: Resultado de identificação *on-line* e Controlo Aditivo Baseado em Modelo Interno (AIMC). A fase inicial de controlo é assegurada por um PI.

```
Verificar leitura e repetir se necessário
Calcular o sinal de controlo:
Usar PI, ou RN se o limiar de qualidade tiver sido atingido
Limitar o sinal de controlo à gama de funcionamento do forno
Aplicar o sinal de controlo
Se i>mínimo_de_pontos
    Repetir (enquanto houver tempo para uma época de treino)
        Utilizar os pontos disponíveis para o treino
        Se dir_inv=1
            Treinar modelo directo
            Avaliar modelo (sequência de teste)
            Guardar o modelo se for o melhor
            Verificar limiar de qualidade
            Verificar condição de paragem de treino antecipada
            Se necessário gerar pesos aleatórios
            Alterar dir_inv se necessário
        caso contrário
            Treinar modelo inverso
            Avaliar modelo (simulação DIC)
```

```
                Guardar o modelo se for o melhor
                Verificar limiar de qualidade
                Verificar condição de paragem de treino antecipada
                Se necessário gerar pesos aleatórios
                Alterar dir_inv se necessário e
                Assinalar início do controlo com a RN
            fim
        fim
    fim
    Aguardar pelo momento exacto de amostragem
    Fazer amostragem
    Guardar resultados
fim
```

As figuras 7.16, 7.17 e 7.18 mostram os resultados de identificação *on-line* e controlo com as seguintes estratégias de controlo: Controlo com Modelo Inverso, Controlo Baseado em Modelo Interno e Controlo Aditivo Baseado em Modelo Interno, utilizando agora ambas as janelas deslizantes.

Neste caso o treino inicia-se após a recolha de 240 amostras, sendo a janela de treino composta por um máximo de 200 amostras e a janela de teste fixa com 100 amostras. Deve notar-se que em relação ao caso anterior ambas as janelas são agora menores para compensar o aumento de esforço computacional que será necessário. O treino é iniciado mais tarde uma vez que são necessárias amostras para a sequência de teste (cujo número de amostras é fixo para não causar problemas adicionais) e para a sequência de treino. As restantes características da implementação são comuns ao caso anterior.

Como foi explicado, nesta segunda situação é usada uma janela de teste, também deslizante. Neste caso tornou-se mais complexa a avaliação de situações onde era conveniente proceder à Paragem de Treino Antecipada e ainda que após cada época de treino se procedesse à comparação entre o melhor modelo e o modelo actual, com o objectivo de guardar o melhor, esta comparação tornou-se também mais complexa uma vez que ao mudar a janela de teste os índices de qualidade dos modelos anteriormente optimizados deixam de ser válidos.

Nesta situação optou-se por, após cada amostragem (e consequente alteração da janela de teste e de treino), se proceder à reavaliação do melhor modelo, garantindo que os modelos são comparados em situações idênticas.

Como é possível verificar pelos resultados apresentados na tabela 7.4, obtiveram-se modelos com qualidade semelhante aos anteriores, com a janela de teste fixa. Com estes modelos não foram encontradas dificuldades em implementar diversas estratégias de controlo.

Estes resultados devem ser analisados de forma cuidadosa, uma vez que, devido à duração das experiências, surgem sempre diversos *outliers* que chegam a ser mais

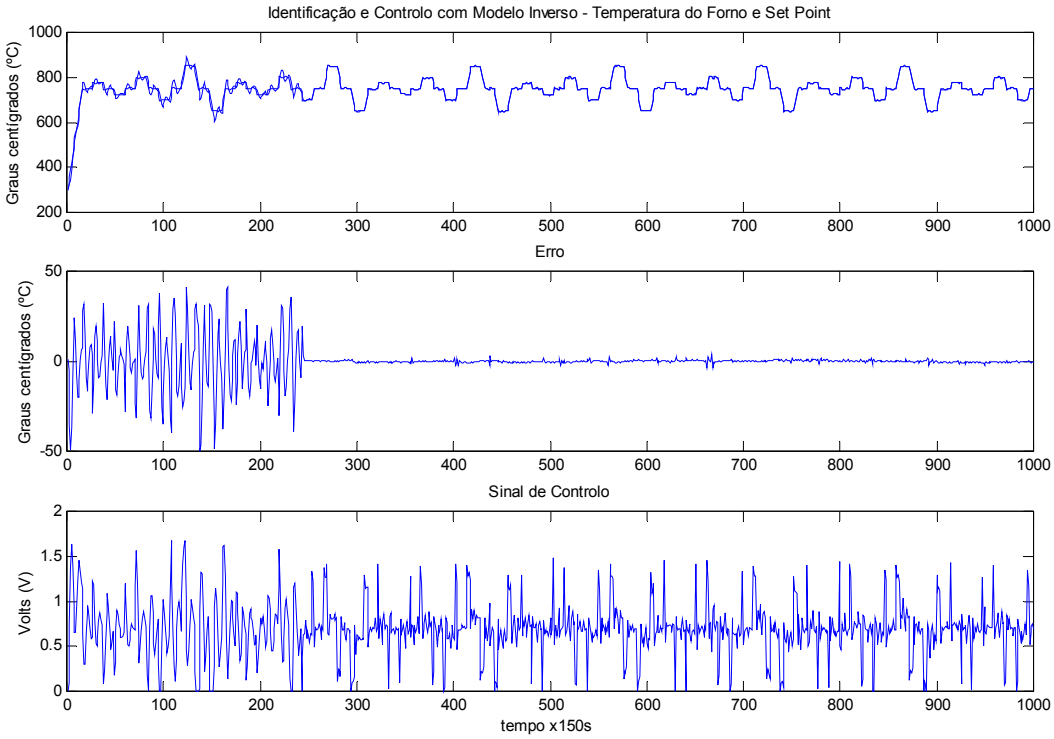


Figura 7.16: Resultado de identificação *on-line* e Controlo com Modelo Inverso (DIC), utilizando janela de teste deslizante. A fase inicial de controlo é assegurada por um PI.

Estratégia\EQM	Amostras 300 a 1000
DIC	0,46
IMC	0,65
AIMC	0,29

Tabela 7.4: Sumário dos valores de EQM para os testes de identificação e controlo on-line, com janela de teste deslizante.

determinantes no valor do EQM do que a própria estratégia de controlo. Estes *outliers* reflectem-se nos valores comparativos apresentados na tabela 7.4.

Apesar desta limitação é possível afirmar que, embora este sinal de referência seja muito mais activo do que o apresentado para os modelos treinados *off-line*, o que naturalmente leva a um EQM mais elevado, o erro obtido com todas as malhas é da mesma ordem de grandeza do erro obtido com os modelos treinados *off-line* e só por esse facto é possível afirmar que estes resultados são muito bons.

No caso da malha IMC, os valores obtidos são os mais elevados para as três estruturas de controlo. Esta situação deve-se essencialmente à existência de *outliers*.

Tal como para o caso da janela de teste fixa, optou-se por apresentar, de forma muito resumida, o algoritmo para facilitar a compreensão da implementação utilizada.

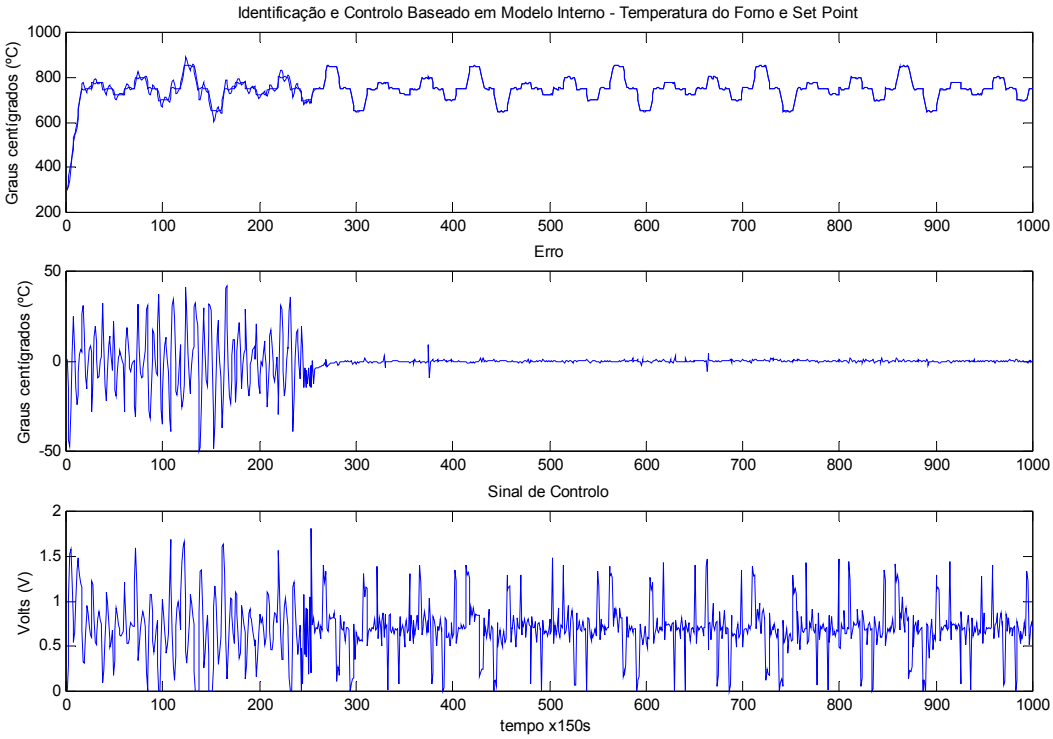


Figura 7.17: Resultado de identificação *on-line* e Controlo Baseado em Modelo Interno (IMC), utilizando janela de teste deslizante. A fase inicial de controlo é assegurada por um PI.

```
Carregar o sinal de referência
Inicializar a comunicação via porta série
Fazer o aquecimento inicial do forno (até aos 300°C)
Configurar os modelos iniciais (Directo e Inverso)
Gerar pesos aleatórios
Desde i=1 até número_de_iterações
    Fazer leitura de temperatura
    Verificar leitura e repetir se necessário
    Calcular o sinal de controlo:
    usar PI, ou RN se o limiar de qualidade tiver sido atingido
    Limitar o sinal de controlo à gama de funcionamento do forno
    Aplicar o sinal de controlo
    Se i>mínimo_de_pontos
        Se existirem:
            Re-avaliar os modelos directo e inverso
            (com a nova janela de teste)
            Substituir os melhores modelos
        fim
```

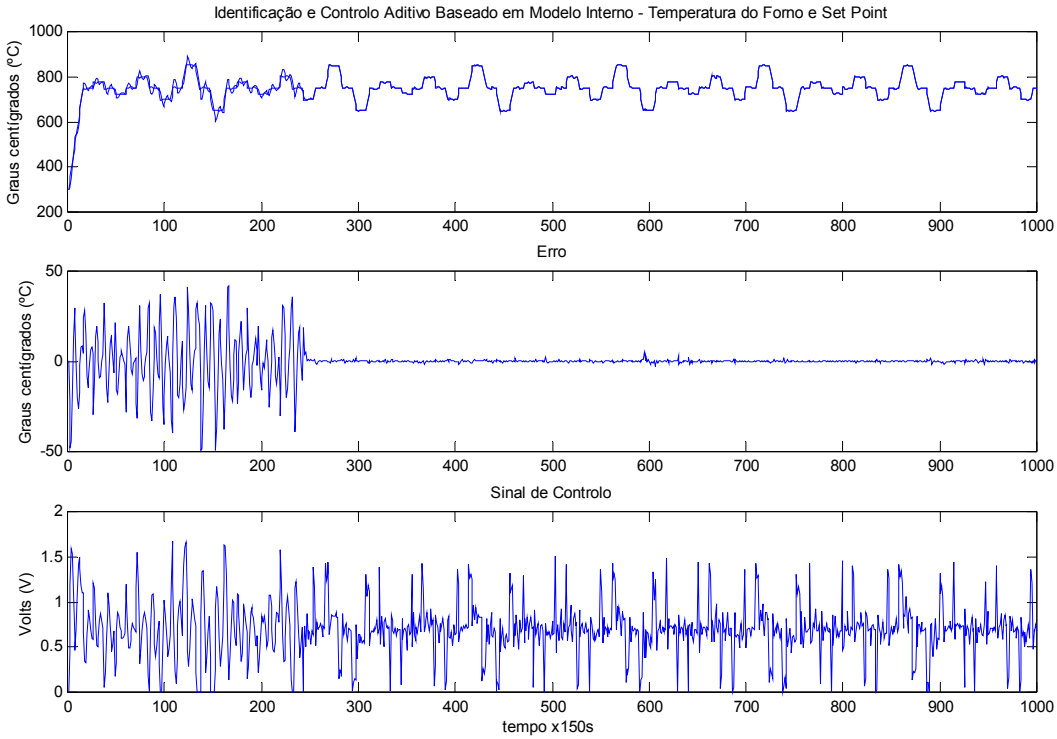


Figura 7.18: Resultado de identificação *on-line* e Controlo Aditivo Baseado em Modelo Interno (AIMC), utilizando janela de teste deslizante. A fase inicial de controlo é assegurada por um PI.

```
Repetir (enquanto houver tempo para uma época de treino)
    Utilizar os pontos disponíveis para o treino
    Se dir_inv=1
        Treinar modelo directo
        Avaliar modelo
        Guardar  o modelo se for o melhor
        Verificar limiar de qualidade
        Verificar condição de paragem de treino antecipada
        Se necessário gerar pesos aleatórios
        Alterar dir_inv se necessário
    caso contrário
        Treinar modelo inverso
        Avaliar modelo
        Guardar  o modelo se for o melhor
        Verificar limiar de qualidade
        Verificar condição de paragem de treino antecipada
        Se necessário gerar pesos aleatórios
        Alterar dir_inv se necessário e
```

```
                Assinalar início do controlo com a RN
            fim
        fim
    fim
    Aguardar pelo momento exacto de amostragem
    Fazer amostragem
    Guardar resultados
fim
```

Como é possível verificar, em termos de algoritmo, a diferença entre as duas versões é mínima. No entanto, em termos de funcionamento a diferença é considerável, uma vez que após cada amostragem é necessário actualizar ambas as janelas deslizantes e em função disso re-avaliar os modelos inversos e directos.

7.6 Outros trabalhos

Vale a pena referir que para além do trabalho descrito nesta tese, foi desenvolvida uma comparação entre sistemas *neuro-fuzzy* e sistemas neuronais. Neste trabalho publicado em [100] e [101]) foram comparados os modelos (no que diz respeito à sua complexidade) e à qualidade de controlo obtida para este mesmo caso prático com as duas soluções referidas.

7.7 Conclusões

Os resultados de controlo com modelos treinados *off-line* e *on-line* atingiram a qualidade que foi imposta nos pressupostos iniciais do trabalho.

A malha AIMC demonstrou a sua utilidade nos processos de treino *on-line* e como malha de controlo.

Apesar de o PC utilizado não ser hoje um modelo que se possa considerar muito actual (como foi descrito na secção 6.2), devido ao intervalo de amostragem ser bastante longo, é possível implementar o algoritmo de Levenberg-Marquardt *on-line*, em janela deslizante, sem um grande acréscimo de complexidade, com a ajuda de um controlador do tipo PI. Isto permite fazer a identificação de um sistema com uma aproximação de caixa preta (do inglês *black box*), uma vez que não é pressuposto nenhum conhecimento prévio sobre o sistema a modelizar e obter modelos de qualidade idêntica aos modelos treinados *off-line*.

Deve notar-se que a janela de treino é consideravelmente mais pequena (em termos de número de amostras) do que o que estava disponível no treino *off-line*. Para se obterem modelos de qualidade aceitável foi preciso realizar mais épocas de treino *on-line* do que as que habitualmente eram necessárias no treino *off-line*. Verificou-se mesmo que o produto número de amostras vezes número de épocas utilizado *on-line* e *off-line* era aproximadamente constante.

De forma mais genérica é possível afirmar que desde que o período de amostragem permita a realização de uma época de treino completa entre duas amostragens, é possível implementar o algoritmo de Levenberg-Marquardt, em janela deslizante. No caso de tal não ser possível, poderá optar-se pela sugestão feita na proposta de trabalho futuro (ver capítulo 9). É portanto possível extrapolar o treino *off-line* para treino *on-line*.

Vale a pena notar também que a qualidade dos modelos obtidos *on-line* é semelhante aos modelos iniciais optimizados pelo operador humano.

Capítulo 8

Implementação de Redes Neuronais em *hardware*

"He who moves not forward goes backward!" - Johann Wolfgang von Goethe, poeta (1749 - 1832)

8.1 Introdução

A implementação mais comum de RNs consiste na utilização de um computador e no desenvolvimento de *software* para a simulação da RN.

Esta forma de implementação, sendo simples, não explora todas as potencialidades de uma RN, sendo habitualmente procurada uma implementação em *hardware* pelos seguintes motivos:

- redução do custo da implementação.
- procura de uma velocidade de processamento mais elevada.
- procura de implementações mais fiáveis.

O presente capítulo é composto por dois artigos, cujos originais em inglês foram publicados em conferências internacionais e que resumem o trabalho desenvolvido em relação a este tema. Os artigos foram traduzidos para português e correspondem quase exactamente aos originais em inglês.

Neste campo, o trabalho inicial feito no âmbito desta tese, foi estudar as soluções de *hardware* disponíveis no mercado, uma vez que estas seriam opções que evitariam um período longo de desenvolvimento de uma solução de raiz. Em resultado desta primeira fase fez-se um estudo das possibilidades existentes no mercado que resultou na publicação dos artigos "*Hardware* Comercial para Redes Neuronais Artificiais: um Estudo" (o título original em inglês foi "*Commercial Hardware for Artificial Neural Networks: a Survey*"), publicado na conferência 5^{th} *IFAC International Symposium on Inteligent Components and Instruments for Control Applications* [102] e "Redes Neuronais Artificiais: uma Revisão do *Hardware* Comercial" (o título original em inglês

é: "*Artificial Neural Networks: a Review of Commercial Hardware*"), publicado na revista *Engineering Applications of Artificial Intelligence* do IFAC [103].

Verificou-se após este estudo que as soluções disponíveis não correspondiam às necessidades existentes para a implementação em *hardware*, nomeadamente no que diz respeito à resolução e à função de activação da RN, que permitissem a utilização directa (sem nenhum tipo de transformação ou reajuste) dos pesos produzidos por um ambiente de treino da RN, em *software.*

Em face da inexistência de uma solução adequada, foi desenvolvida uma solução de raiz, tendo sido escolhida como plataforma para implementação uma FPGA (*Field Programmable Gate Array*) devido principalmente ao custo menor para um baixo número de cópias produzidas. A apresentação da solução desenvolvida foi feita através do artigo "Processador de Redes Neuronais Artificiais - uma Implementação em *Hardware* Usando uma FPGA" (o título original em inglês foi "*Artificial Neural Networks Processor - a Hardware Implementation Using a FPGA"*) publicado na conferência *Field-Programmable Logic and its Applications* [104].

Desta solução há a realçar o processo de implementação da função de activação que, sem ser demasiado dispendioso em termos de recursos, permite obter uma resolução elevada e também a elevada precisão obtida para a implementação completa que permite a utilização directa de pesos produzidos em ambientes de elevada resolução.

Os modelos utilizados nesta parte do trabalho são modelos obtidos com período de amostragem de 30s.

8.2 *Hardware* Comercial

Título: *Hardware* comercial para redes neuronais artificiais: um estudo

Autores: Fernando Morgado Dias, Ana Antunes e Alexandre Manuel Mota

8.2.1 Sumário

As Redes Neuronais transformaram-se numa solução comum para uma grande variedade de problemas em muitas áreas, tais como o controlo e o reconhecimento de padrões, para referir apenas alguns casos. Muitas soluções encontradas nestas e noutras áreas das Redes Neuronais alcançaram uma fase de implementação em *hardware*, quer seja a partir de soluções comerciais ou com base em protótipos. A solução mais frequente para a implementação de redes neuronais consiste em treinar e implementar as redes dentro de um computador. Não obstante, esta solução pode não ser apropriada por causa do seu custo ou da sua velocidade limitada. A implementação pode ser demasiado cara por causa do computador e demasiado lenta quando executada em *software.* Em ambos os casos o *hardware* dedicado pode ser uma solução interessante.

A necessidade de *hardware* dedicado pode não implicar construir uma nova solução, uma vez que nas últimas duas décadas diversas soluções comerciais de *hardware* que podem ser usadas na implementação foram colocadas no mercado.

Infelizmente nem todos os circuitos integrados corresponderão às necessidades: alguns utilizam uma precisão demasiado baixa, outros implementam apenas determinado tipo de redes, outras ainda não têm o treino incluído e a informação não é fácil de encontrar.

Este artigo está confinado a relatar apenas as soluções comerciais que foram desenvolvidas especificamente para redes neuronais, deixando de fora outras soluções.

Esta opção foi feita porque a maioria das outras soluções são baseadas em placas de circuito impresso que são construídas com estes circuitos integrados ou com processadores digitais de sinal (do inglês *Digital Signal Processor* - DSP) e processadores de conjunto de instruções reduzido (*Reduced Instruction Set Computers* - RISC).

Palavras Chave: Redes Neuronais Artificiais, *Hardware*.

8.2.2 Introdução

A área das Redes Neuronais (RNs) atravessou diferentes fases de desenvolvimento. Uma das etapas mais importantes foi conseguida quando Cybenko [9] provou que poderiam ser usadas como aproximadores universais.

Uma fase negativa surgiu no livro de Minsky e Papert chamado Perceptrões [105]. Esta fase negativa foi superada quando foram propostos algoritmos para o treino de RNs com diversas camadas nos anos 80. Desde então muito trabalho tem sido feito a respeito das RNs e da sua aplicação a muitas áreas diferentes [106]. Naturalmente a aplicação bem sucedida a algumas áreas conduziu a aplicações comerciais ou específicas que podem trabalhar sem necessitar de um computador.

A necessidade de abandonar a implementação mais comum das RNs, com um computador, pode levantar-se por diversas razões: reduzir o custo da implementação, conseguir velocidades de processamento mais elevadas ou implementações mais simples.

Reduzir o custo ou ter implementações mais simples são objectivos que podem ser conseguidos simplesmente substituindo o computador por *hardware* específico.

Ao contrário da arquitectura dos computadores convencionais (arquitectura de von-Neumann) que é sequencial, as RNs beneficiam da capacidade de fazer processamento paralelo maciço [107]. Isto pode ser explorado pelo *hardware* específico para aumentar a velocidade de processamento.

Para estas aplicações, que compartilham a necessidade de trabalhar sem um computador, algum *hardware* dedicado tem sido construído, o que pode evitar a dificuldade de produzir *hardware* para cada nova aplicação.

O *hardware* produzido foi resultado de diferentes necessidades e tem consequentemente usos diferentes.

Para escolher o *hardware* para uma aplicação específica são necessários os detalhes de cada circuito.

As diferentes soluções poderão ser úteis ou não dependendo da precisão usada para os pesos, do número máximo de pesos, do tipo de rede implementado, da disponibilidade de algoritmos de treino no circuito e de outras características.

Este artigo está confinado aos circuitos comerciais que foram desenvolvidos especificamente para RNs, independentemente da tecnologia usada: circuitos integrados es-

pecíficos para a aplicação (do inglês *Application Specific Integrated Circuits* - ASICs), FPGAs, *Sea of Gates* ou outro, excluindo outras soluções. Esta opção foi feita porque, com excepção de algumas propostas híbridas, a maioria das outras soluções são baseadas em placas de circuito impresso que são construídas com base nestes circuitos, DSPs ou processadores RISC.

A utilidade deste estudo pode, consequentemente, ser sumariada em dois sentidos diferentes: uma referência resumida para aqueles que necessitam do *hardware* para uma implementação específica e informação sobre as soluções existentes para quem procura desenvolver uma nova implementação.

8.2.3 Especificação do *hardware*

Do ponto de vista do utilizador, a primeira etapa para escolher uma solução de *hardware* é uma especificação resumida do que é pretendido. Esta especificação inclui o tipo de RN (FNN, RBF, Kohonen, etc.), o número dos neurónios, o número de entradas e de saídas, o número de ligações a cada neurónio, a precisão, a velocidade de operação ou o desempenho e outras características que podem ser mais ou mais menos importantes dependendo da aplicação.

A maioria destas características podem ser retiradas directamente da aplicação, enquanto outras merecerem ser analisadas com maior detalhe.

A precisão usada é um parâmetro importante, até porque pode existir uma precisão diferente para diferentes partes: entradas, saídas, pesos, cálculos internos, acumuladores e multiplicadores.

O desempenho do circuito pode ser medido de muitas formas diferentes e é um assunto que está longe de ser consensual.

A avaliação de desempenho mais comum é o número de ligações por segundo (do inglês *Connection Per Second* - CPS) [108] que é definido como o número de operações de multiplicação e acumulação efectuadas por segundo durante a fase de processamento das saídas da RN. Uma medida equivalente existe para a fase de treino: número de actualizações das ligações por segundo (do inglês *Connection Update Per Second* - CUPS) e mede o número de mudanças nos pesos por segundo.

Existem igualmente outras medidas. O valor do CPS pode ser normalizado através da divisão pelo número dos pesos N_w (equação 8.1) e obtém-se o número de ligações por segundo e por peso (do inglês *Connection Per Second Per Weight* - CPSPW), que foi sugerido como um critério melhor para avaliar o desempenho de cada solução [109].

$$CPSPW = CPS/N_w \tag{8.1}$$

Uma outra medida é o número de ligações primitivas por segundo (do inglês *Connection Primitives Per Second* - CPPS), que pode ser calculado da seguinte forma:

$$CPPS = b_{in}.b_w.CPS \tag{8.2}$$

onde b_{in} é o número de bits usados para as entradas e b_w é o número de bits usados para os pesos. Esta medida permite que a precisão seja incluída na medida de

desempenho [110] [111]. Estas duas medidas podem também ser aplicadas à fase de treino, sendo usada a medida CUPS em vez de CPS.

Um outro parâmetro que pode ser usado para analisar o desempenho é a dissipação de potência [111]. Dependendo da tecnologia, da frequência do sinal de relógio, do número de elementos de processamento, da precisão, etc., cada solução de *hardware* tem uma medida de dissipação de potência que não pode ser comparada directamente. Uma medida de energia por ligação foi proposta por [111]. Esta medida é um indicador da eficiência energética dos circuitos e está a tornar-se mais importante à medida que a integração das soluções aumenta porque a necessidade de dissipar a potência limita a capacidade de integração ao nível de sistema.

Infelizmente esta medida não está frequentemente disponível para a maioria dos circuitos, o que tornou impossível a sua inclusão neste estudo.

Para uma especificação mais detalhada outra informação pode ser tomada em conta: facilidades de treino, facilidade de utilizar diversos circuitos do mesmo tipo em conjunto, tipo de armazenamento dos pesos, tipo de implementação das funções da activação, frequência de relógio, número de entradas e saídas e tecnologia ou tipo de implementação do circuito (analógico, digital ou misto).

8.2.4 Tipos de implementação

Classificar os tipos de implementação das soluções de *hardware* é uma tarefa controversa, como pode ser visto nos exemplos presentes na literatura. Por exemplo em [112] é proposta uma classificação que divide as soluções de *hardware* em Neuro-*chips* e Neurocomputadores que têm ambos as subcategorias de gerais e especializados. Uma outra classificação, embora mais geral, uma vez que se destina a componentes de *hardware* é apresentada em [113], na qual as classes usadas são: neuro-computadores, placas aceleradoras, *chips*, elementos de biblioteca e micro-computadores encaixados (do inglês *embedded microcomputers*).

Em [114] a classificação é feita de acordo com a figura 8.1. Aqui a solução global de *hardware* é chamada Neuro-computadores, que podem ser *chips* standard ou Neuro-*chips*. Os *chips* standard podem ser classificados como sequenciais e placas aceleradoras ou soluções multi-processador.

Os Neuro-*chips* são ASICs e podem ser classificados como analógicos, digitais ou híbridos.

O presente artigo trata somente da categoria de Neuro-*chips* com ênfase especial para as soluções comercialmente disponíveis.

A classificação usada aqui é similar ao proposto em [114] para Neuro-*chips*, uma vez que usa uma separação das soluções de acordo com sua natureza: analógicas, digitais ou híbridas.

8.2.5 *Hardware* comercial

De acordo com a classificação simplificada previamente definida o *hardware* existente será agora apresentado.

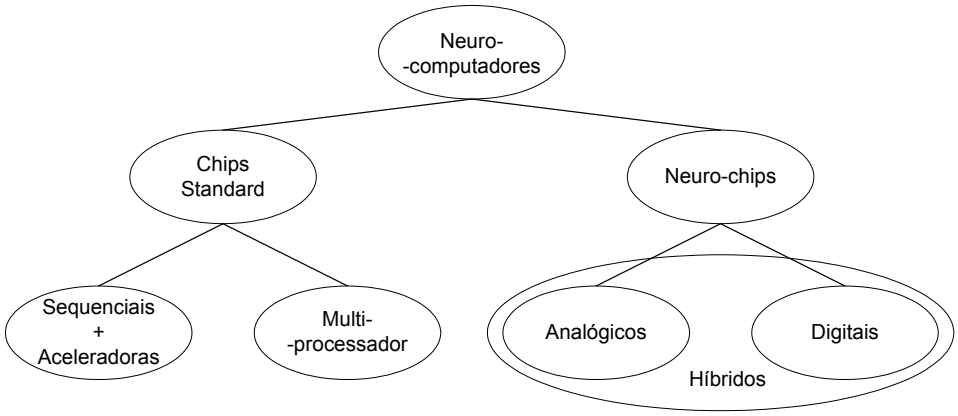


Figura 8.1: Classificação do *hardware* para RNs em categorias.

As soluções encontradas são apresentadas com informação sobre a arquitectura, a capacidade implementada para aprender, a precisão, o número de neurónios disponíveis, o número de sinapses ou a memória disponível para as implementar e uma medida da velocidade de operação ou de desempenho.

Deve notar-se que algum do *hardware* apresentado não está actualmente em produção mas é incluído ainda como referência porque pode estar disponível em alguns fornecedores.

Para cada elemento apresentado na tabela a informação é detalhada no texto e as abreviaturas explicadas.

Arquitectura: Neste tópico é apresentada a informação referente ao tipo de rede que pode ser implementado e, em alguns casos, é também fornecida informação adicional sobre o tipo de implementação.

Os exemplos são: FNN, Circuitos que implementam operações de matrizes, etc.

As abreviaturas usadas são: utilização geral (do inglês *General Purpose*) (GP), instrução única para vários dados (do inglês *Single Instruction Multiple Data* - SIMD), vírgula flutuante (do inglês *Floating Point* - FP), inteiro (Int), RBF, com todas as ligações e realimentado (do inglês *Fully Connected and Recurrent* - FCR).

Treino: Neste tópico é fornecida a informação sobre a possibilidade de treino dentro do circuito integrado.

Os exemplos são: Programa, quando o algoritmo a usar puder ser programado, Hopfield, Boltzmann, etc.

As abreviaturas usadas são: Backpropagation (BP), região de influência (do inglês *Region of Influence* - ROI), energia restringida de Coulomb (do inglês *Restricted Coulomb Energy* - RCE), rede neuronal probabilística (do inglês *Probabilistic Neural Networks* - PNN), k - vizinho mais próximo (do inglês *K-Nearest Neighbour* - KNN), L1 e LSUP (duas normas para serem usadas com redes do tipo RBF [115]).

Precisão: A informação sobre a precisão é apresentada no formato número de bits por entrada vezes o número de bits por saída. A informação sobre a precisão interna raramente está disponível, o que não é muito satisfatório tanto do ponto de

Nome	Arq.	Treino	Prec.	Neur.	Sin.	Velocidade
Intel ETANN	FNN	Não	6bx6b	64	10280	2 GCPS
Synaptics Silicon Retina	Neuro-morphic	Não	N.A.	48x48	Resis-tive net	No

Tabela 8.1: Implementações analógicas de RNs.

vista científico como do ponto de vista industrial.

Neurónios: Este campo indica o número de neurónios disponíveis no circuito. Em alguns casos o circuito contém elementos de processamento (do inglês *Processing Elements* - PE) em vez de neurónios.

Sinapses: Este campo indica o número de ligações disponíveis ou a quantidade de memória disponível para armazenar os valores dos pesos.

Velocidade: Neste campo é fornecida uma medida do desempenho. A maioria das abreviaturas já foram mencionadas na secção 2 e pat/s significa o número de padrões a ser processados em cada segundo (em actividades de reconhecimento de padrões).

Em todos os campos N.A. (do inglês *Not Available*), significa que a informação não está disponível.

A informação apresentada foi recolhida directamente dos fabricantes ou estava disponível através de estudos precedentes sobre *hardware* para RNs.

Implementações analógicas

Em geral, as implementações analógicas têm a vantagem de obter velocidade elevada e alta densidade, ao mesmo tempo que apresentam diversos inconvenientes tecnológicos. Estas dificuldades estão relacionadas principalmente com: armazenamento dos pesos, estabilidade com variações da temperatura e obtenção de multiplicadores lineares sobre uma gama de operação alargada. Naturalmente estes problemas podem resultar numa perda de precisão quando comparados a outros tipos de implementações.

A tabela 8.1 mostra o resumo das características das implementações analógicas.

ETANN - o *chip* da Intel, 80170NX, também designado por rede neuronal analógica treinável electricamente (do inglês *Electrically Trainable Analog Neural Network* - ETANN), foi o primeiro a estar disponível comercialmente. Armazena os pesos como carga eléctrica em portas flutuantes e usa multiplicadores de Gilbert de 4-quadrantes. Tem 64 neurónios e consegue um desempenho de 2 GCPS.

Synaptics Silicon Retina - este *chip* é um caso especial na implementação de *hardware* de RNs porque em vez de implementar uma arquitectura convencional tenta emular os neurónios biológicos da forma mais aproximada possível [108].

Implementações digitais

As implementações digitais, em geral, têm as seguintes vantagens: armazenamento dos pesos em memória, facilidade de integração com outras aplicações, maior facilidade de

implementar os algoritmos de treino e precisão dentro do número de bits dos operandos e dos acumuladores. De uma forma simplificada, quando comparadas com as soluções analógicas, as soluções digitais podem ser consideradas mais precisas.

Por outro lado as implementações digitais são mais lentas (devido à necessidade de conversão de entradas analógicas e à operação dos multiplicadores digitais) e apresentam mais dificuldades para implementar as funções da activação.

As implementações digitais estão habitualmente divididas em quatro classes: arquitecturas *slice*, SIMD, *systolic arrays* [116] [108] e RBFs embora outras classes sejam propostas em alguns artigos, como *chips* multiprocessador ou alternativas que resultam da tecnologia usada (PLD e FPGA) ou da semelhança com o tipo de *hardware* (por exemplo DSPs).

As características de cada classe e a divisão dos circuitos em classes são apresentadas nas tabelas 8.2, 8.3, 8.4, 8.5 e 8.6, que mostram o resumo das características das implementações digitais.

Arquitecturas *Slice*: O princípio das arquitecturas *slice* é o de fornecer blocos para construir RNs de tamanho arbitrário, embora os blocos possam ser usados para implementar uma RN por si próprios. Em muitos casos estes blocos implementam um neurónio e o nome deste tipo de implementação deriva do conceito de *bit slice* dos processadores digitais convencionais [108].

Por causa do tipo de arquitectura utilizado, esta solução geralmente não inclui o treino no próprio circuito.

Os exemplos deste tipo de solução são (as características podem ser vistas na tabela 8.2):

MD-1220[1] - este *chip* da Micro Devices foi uma das primeiras implementações de *hardware* comercial para RNs. Contém oito neurónios com funções da activação de Heaviside e oito pesos de dezasseis bits com apenas um bit de entrada. Os multiplicadores usados são do tipo *bit-serial.*

NLX-420[2] - cada *chip* contém dezasseis neurónios com precisão variável. Os dezasseis bits disponíveis para as entradas e os pesos podem ser seleccionados como dezasseis elementos de um bit, quatro elementos de quatro bits, dois elementos de oito bits ou um de dezasseis bits. As funções de activação são definidas pelo utilizador e está disponível realimentação interna.

Lneuro-1 - composto por dezasseis PEs, este *chip* tem também dezasseis bits de precisão variável para os pesos como o NLX-420. Os pesos devem ser guardados na memória *cache* existente no *chip* (1Kbyte) e as funções de activação são guardadas externamente.

Lneuro-2.3 - o Lneuro 2.3 é uma evolução do Lneuro 1. É constituído por 12 PEs que podem ser usados de forma paralela ou em modo série. Cada PE contém 128 registos de 16 bits que podem ser usados para armazenar os pesos ou os estados. Os PEs contêm também 16 a 32 bits de resolução no multiplicador, uma ALU de 32 bits

[1] Não disponível.

[2] Não disponível.

Nome	Arq.	Treino	Prec.	Neur.	Sin.	Velocidade
Arquitectura *Slice*						
Micro Devices MD-1220	FNN	Não	1bx16b	8	8	1,9MCPS
NeuraLogix NLX-420 Retina	FNN	Não	1-16b	16	Ext.	300CPS
Philips Lneuro-1	FNN	Não	1-16b	16 PE	64	26MCPS
Philips Lneuro-2.3	N.A.	Não	16-32b	12 PE	N.A.	720MCPS

Tabela 8.2: Implementações digitais com arquitectura *slice*.

e um *shift-barrel*. Pode ser usado microcódigo para adaptar o *chip* às aplicações.

Arquitectura SIMD: Tal como o próprio nome explica, cada processador executa a mesma instrução em paralelo sobre dados diferentes.

Os exemplos deste tipo são (as características podem ser vistas na tabela 8.3):

Inova N64000[3] - Este *chip* contém 64 PEs cada um com um multiplicador de inteiros de 9 por 16 bits, um acumulador de 32 bits e 4Kbytes de memória para o armazenamento dos pesos. O circuito inclui linhas de controlo e de dados comuns para cada processador o que torna mais fácil a combinação de diversos *chips*.

HNC100-NAP[4] - O Hecht-Nielsen *Computers* 100 *Neurocomputer Array Processor* é um processador de vírgula flutuante de 32 bits. A escolha de uma representação de vírgula flutuante de 32 bits limita o número de PEs a 4. Os pesos são armazenados exteriormente.

Hitachi WSI - Existem dois circuitos diferentes desenvolvidos pela Hitachi que têm algumas características comuns e algumas características distintas. Em comum têm a precisão de 9 por 8 bits e o tipo de arquitectura. A parte distinta diz respeito ao treino (Hopfield e *Backpropagation* com termo de momentum [118]), ao número dos neurónios e à memória disponível para as sinapses.

Neuricam Nc3001 Totem - o Nc3001 é designado como um processador de sinal paralelo e é composto por 32 DSPs. A arquitectura permite ter 1 a 32 neurónios implementados. A precisão usada é de 32 bits, com facilidades para a implementação das funções de activação em RAM através de *Look Up Table* (LUT). Os multiplicadores usados são do tipo Baugh-Wooley em *pipeline* [119].

Neuricam Nc3003 Totem - o Nc3003 é uma evolução do Nc3001 embora os dois *chips* sejam muito semelhantes. Analisando só as características escolhidas, a diferença é apenas uma memória maior disponível para armazenar os pesos. Há também uma melhoria no controlo interno do circuito [120].

[3]Não disponível de acordo com [117].

[4]Não disponível.

Nome	Arq.	Treino	Prec.	Neur.	Sin.	Velocidade
Arquitectura SIMD						
Inova N64000	GP, Int, SIMD	Program	1-16b	64 PE	256k	870MCPS 220MCUPS
Hecht-Nielson HNC 100-NAP	GP, FP, SIMD	Program	32b	4 PE	512k Ext.	250MCPS 64MCUPS
Hitachi WSI	Wafer, SIMD	Hopfield	9bx8b	576	32k	138MCPS
Hitachi WSI	Wafer, SIMD	BP	9bx8b	144	N.A.	300MCUPS
Neuricam Nc3001 Totem	FNN SIMD	Não	32b	1-32	32k	1GCPS
Neuricam Nc3003 Totem	FNN SIMD	Não	32b	1-32	64k	750MCPS
RC Module NM6403	FNN	Program	1-64bx 1-64b	1-64	1-64	1200MCPS

Tabela 8.3: Implementações digitais com arquitectura SIMD.

Nome	Arq.	Treino	Prec.	Neur.	Sin.	Velocidade
Systolic Array						
Siemens MA-16	Matriz oper.	Não	16b	16 PE	16x16	400MCPS

Tabela 8.4: Implementações digitais com arquitectura *systolic array*.

RC Module NM6403 - este circuito foi projectado pela RC Module e produzido pela Samsung. O circuito é composto por processadores RISC de 32/64 bits e por um co-processador de 64-bit para suportar operações vectoriais de comprimento variável e é uma combinação das arquitecturas de palavra muito longa (do inglês *Very Long Instruction Word* - VLIW) e SIMD [121].

Arquitectura *Systolic Array*: Em soluções do tipo *systolic array*, cada processador executa uma etapa do cálculo (sempre a mesma etapa) antes de passar o resultado ao processador seguinte, dando origem a um processamento do tipo *pipeline*.

O único exemplo deste tipo é (as características podem ser vistas na tabela 8.4):

Siemens MA-16 - Este *chip* executa operações rápidas de matrizes 4x4 com elementos de 16 bits. Tem 16 PEs complexos [118] e os pesos e as funções da activação são armazenados exteriormente (estes através de LUT).

Arquitectura RBF: Este tipo de RNs requer geralmente *hardware* específico, embora não existam muitas soluções comerciais disponíveis.

Nome	Arq.	Treino	Prec.	Neur.	Sin.	Velocidade
Arquitectura RBF						
Nestor/ Intel NI1000	RBF	RCE, PNN, program	5b	1 PE	256x 1024	40kpat/s
IBM ZISC036	RBF	ROI	8b	36	64x36	250kpat/s
Silicon Recognition ZISC 78	RBF	KNN, L1, LSUP	N.A.	78	N.A.	1Mpat/s

Tabela 8.5: Implementações digitais com arquitectura RBF.

Exemplos deste tipo são (as características podem ser vistas na tabela 8.5):

Nestor/Intel Ni 1000[5] - este circuito desenvolvido em conjunto pela Nestor e pela Intel, pode armazenar até 1024 protótipos com 256 dimensões e 5 bits por dimensão. Os algoritmos de RCE e de PNN estão disponíveis para treino *on-line*, com a possibilidade de usar qualquer outro algoritmo através de microcódigo. De acordo com [122] este *chip* foi retirado do mercado pela Intel mas continuará a ser produzido para a Nestor sob um acordo de 15 anos.

IBM ZISC036 - O IBM *Zero Instruction Set Computer* contém 36 neurónios com uma precisão de 8 bits. O *chip* tem treino integrado para o algoritmo ROI e inclui facilidades para ligar em série vários *chips*.

Silicon Recognition ZISC 78 - este circuito está preparado para fazer reconhecimento automático de padrões através da utilização do algoritmo KNN e das normas L1 e LSUP e é composto por 78 neurónios. Os vectores da entrada podem ter 1 a 64 componentes de 8 bits [115].

Outros *chips*: Alguns dos circuitos desenvolvidos não cabem nas classes acima por causa das suas características. Um exemplo é o circuito SAND que pode ser classificado como um *systolic array* ou um RBF (as características podem ser vistas na tabela 8.6).

SAND/1 - o dispositivo *Simply-Applicable Neural Device* é uma solução projectada com uma tecnologia de *Sea of Gates* que poderia ser colocada em diversas classes. É constituída por 4 PEs, pesos armazenados externamente e funções de activação com 40 bits. O SAND necessita também de um circuito sequenciador adicional para gerir a memória e efectuar o controlo [123].

MCE MT19003 - este circuito é classificado como um *Neural Instruction Set Processor* – NISP, ou seja um RISC muito simples com somente sete instruções optimizadas para a implementação de FNN.

Implementações Híbridas

As implementações híbridas aparecem como um meio para tentar obter o melhor dos sistemas analógicos e dos sistemas digitais. Tipicamente as entradas/saídas são digitais

[5] Não disponível de acordo com [122]

Nome	Arq.	Treino	Prec.	Neur.	Sin.	Velocidade
Outros circuitos						
SAND/1	FNN, RBF, Kohonen	Não	40b	4 PE	Ext.	200MCPS
MCE MT19003	FNN	Não	13b	8	Ext.	32MCPS

Tabela 8.6: Implementações digitais com outras arquitecturas.

para facilitar a integração em sistemas digitais, enquanto internamente algum ou todo o processamento é analógico [108].

A tabela 8.7 mostra o resumo das características das implementações híbridas.

AT&T ANNA - AT&T *Artificial Neural Network* ALU (*Arithmetic Logic Unit*) é um circuito digital do ponto de vista do utilizador mas é analógico no seu interior. Os pesos são armazenados como carga de condensadores que é refrescada periodicamente. Este *chip* fornece um número variável de neurónios (16 a 256) e não tem nenhum algoritmo de treino incluído.

Bellcore CLNN -32 – este circuito apresenta semelhanças com o ANNA, uma vez que as entradas são também digitais e o processamento interno é analógico. Neste caso o treino com o algoritmo de Boltzmann está disponível internamente. Implementa redes realimentadas com todas as ligações (do inglês *fully connected recurrent networks*) [124].

Mesa Research Neuroclassifier - este circuito, construído pelo instituto de pesquisa Mesa da Universidade de Twente, tem entradas analógicas e saídas com pesos digitais de 5 bits. A velocidade que é reivindicada atinge os 21 GCPS, que é a taxa de desempenho mais elevada encontrada nas soluções comerciais.

Ricoh RN-200 - este *chip* implementa um método diferente de emular RNs, baseado na taxa de impulsos e na largura dos impulsos [108]. Esta é uma evolução do Ricoh RN-100, que tinha treino disponível com *Backpropagation* modificado. Um conjunto de 12 Ricoh RN-100 foi usado para aprender como balançar um pêndulo 2-D em apenas 30 segundos [108]. Esta versão tem 16 neurónios cada um com 16 sinapses e consegue um desempenho de 3GCPS.

8.2.6 Dificuldades encontradas neste trabalho

Vale a pena apontar as dificuldades encontradas para desenvolver este trabalho porque elas estão relacionadas com a sua utilidade.

O problema principal teve a ver com a obtenção da informação a partir dos fabricantes. Embora possa parecer estranho e seja talvez o resultado destes produtos não terem grande expressão na venda a retalho, a maioria dos inquéritos feitos aos fabricantes não obtiveram resposta.

Isto deu origem a que alguma da informação colocada neste artigo seja proveniente de outros artigos, em vez de ser proveniente do fornecedor.

Nome	Arq.	Treino	Prec.	Neur.	Sin.	Velocidade
AT&T ANNA	FNN	Não	3bx6b	16-256	4096	2,1GCPS
Bellcore CLNN-32	FCR	Boltzmann	6bx5b	32	992	100MCPS 100MCUPS
Mesa Research Neural-classifier	FNN	Não	6bx5b	6	426	21GCPS
Ricoh RN-200	FNN	BP	N.A.	16	256	3GCPS

Tabela 8.7: Implementações analógicas de RNs.

Devido a esta dificuldade, os autores foram incapazes de verificar se todos os circuitos estão ainda disponíveis e de efectuar uma validação de toda a informação usando os datasheets dos produtos.

Alguns *chips* incluídos neste artigo são incluídos apenas como referência uma vez que já não estão disponíveis a partir do fabricante. Não obstante, poderá ser possível encontrá-los junto de alguns fornecedores.

Por causa da dificuldade para recolher a informação destes produtos, alguns circuitos mais recentes não foram incluídos, uma vez que não foi possível encontrar informação suficiente para os classificar de forma a serem incluídos.

Por outro lado, as dificuldades relatadas aqui fazem com que este tipo de artigos se tornem mesmo mais importantes para qualquer um que comece a procurar informação neste campo, uma vez que seria necessário um grande esforço para recolher toda esta informação.

8.2.7 Conclusões

Alguns novos neuro-*chips* foram apresentados neste estudo enquanto vários outros deixaram de estar disponíveis comercialmente.

As soluções novas que apareceram indicam que este campo está ainda activo, mas a remoção do mercado de algumas soluções não é uma boa notícia.

Como [114] indica, a construção de neuro-computadores é cara, em termos de tempo e dos recursos necessários e pouco é conhecido sobre as perspectivas comerciais para as novas implementações. Além disso, não há um consenso evidente na forma de explorar as potencialidades tecnológicas actualmente disponíveis no VLSI (*Very Large Scale Integration*) e mesmo no ULSI (*Ultra Large Scale Integration*) para a implementação de *hardware* massivamente paralelo para RNs. Uma outra razão para que não se estejam a desenvolver novos neuro-computadores pode residir no facto de o número de novos paradigmas de RNs estar a aumentar rapidamente. Para muitos desses novos paradigmas as potencialidades são ainda mal conhecidas. Paradoxalmente, estas potencialidades só podem ser testadas completamente quando o *hardware* dedicado estiver disponível.

Estas poderão ser as razões para o desenvolvimento lento do mercado de *hardware* de RNs nos últimos anos, mas os autores acreditam que esta situação mudará no futuro próximo com o surgimento de novas soluções de *hardware*.

Na perspectiva do utilizador, analisando a informação fornecida aqui sobre o que existe no mercado, deve perceber-se que não há uma escolha ideal, mas sim uma solução mais adequada para cada caso. Esta é a razão que levou os autores a não fazer uma comparação de desempenho.

8.3 Implementação usando uma FPGA

Título: Processador de Redes Neuronais Artificiais - uma implementação de *hardware* usando uma FPGA

Autores: Pedro Ferreira, Pedro Ribeiro, Ana Antunes, Fernando Morgado Dias

8.3.1 Sumário

Diversas implementações de redes neuronais artificiais em *hardware* têm sido apresentadas em artigos científicos. Apesar disso, estas implementações não permitem o uso directo das redes treinadas *off-line* por causa da precisão usada, que é muito mais baixa quando comparada com as soluções de *software* onde os modelos são preparados, ou por causa das modificações na função de activação. No trabalho actual foi desenvolvida uma solução de *hardware* chamada Processador de Redes Neuronais Artificiais usando uma FPGA, que corresponde às exigências para uma implementação directa das redes neuronais sem realimentação, devido à elevada resolução e ao grau de exactidão da função de activação que foram obtidos. A solução de *hardware* resultante é testada com dados de um sistema real para confirmar que pode implementar correctamente os modelos preparados *off-line* com o MATLAB.

Palavras Chave: FPGA, Redes Neuronais Artificiais, *Hardware*, VHDL.

8.3.2 Introdução

As redes neuronais artificiais transformaram-se numa solução comum para uma grande variedade de problemas em muitas áreas, tais como o controlo e reconhecimento de padrões, para referir apenas alguns casos. Dentro destas soluções, uma grande parte foi desenvolvida usando um tipo de RNs que permite apenas ligações no sentido da saída (isto é, entre uma camada e a seguinte mais perto da saída), chamado redes neuronais sem realimentação (do inglês *Feedforward Neural Networks* - FNN). Consequentemente não é uma surpresa que algumas das soluções tenham alcançado uma fase de implementação onde o *hardware* específico seja considerado uma alternativa melhor do que a implementação mais comum usando um computador pessoal (do inglês *Personal Computer* - PC) ou uma estação de trabalho.

Várias razões podem ser apontadas como motivo para esta escolha:

- Necessidade de velocidade de processamento mais elevada.

- Redução do custo para cada implementação.
- Fiabilidade.

O PC ou a estação de trabalho sendo arquitecturas convencionais de von-Neumann não podem fornecer a velocidade de processamento elevada que é requerida por muitas aplicações. Uma solução de *hardware* pode também ser menos cara e mais fiável do que um PC.

Como indicado em [125] "O maior potencial das redes neuronais permanece na elevada velocidade de processamento que poderia ser fornecida com as implementações VLSI massivamente paralelas".

Dentro das soluções (analógico, digital ou híbrido) que é possível encontrar entre as implementações comerciais [102], as digitais têm as seguintes vantagens:

- Armazenamento dos pesos em memória.
- Facilidade da integração com outras aplicações.
- Facilidade de implementação dos algoritmos de treino.
- Exactidão dentro do número de bits dos operandos e dos acumuladores.
- Baixa sensibilidade ao ruído eléctrico e à temperatura.

Considerando as soluções possíveis para uma implementação digital, há ainda escolhas a fazer: *full custom* ASICs, *Sea of Gates*, FPGAs (para referir apenas as soluções que já têm sido usadas na implementação de RNs). A escolha real está de facto bastante reduzida uma vez que para a maioria das aplicações deve ser projectada uma solução específica de *hardware* e consequentemente é impossível usar o mesmo projecto para muitas aplicações, o que torna economicamente inviável a utilização de ASICs ou de soluções com a tecnologia *Sea of Gates*.

Isto conduz, no caso de uma aplicação específica, a uma solução com uma FPGA porque o custo associado à produção de apenas algumas cópias do *hardware* é aceitável. Na literatura é possível verificar que diversas soluções têm já sido testadas no contexto das FPGA. Apesar disso, as soluções que foram encontradas não permitem o uso directo dos modelos neuronais que são preparados normalmente com *software* (como o MATLAB ou *software* específico para RNs) dentro de PCs ou de estações de trabalho.

Todas as soluções que os autores puderam verificar apresentam uma precisão muito mais baixa do que a disponível com as soluções de *software* [126], [127] ou as modificações na função de activação tornam-nas inaceitáveis para usar directamente os pesos previamente preparados [128], [129].

Os autores acreditam que seria do maior interesse que fossem desenvolvidas soluções de *hardware* que possam usar directamente os pesos preparados em *software*.

No trabalho actual, é apresentada uma solução de *hardware* chamada processador de redes neuronais artificiais (do inglês *Artificial Neural Network Processor* - ANNP), usando uma FPGA, que corresponde às exigências acima descritas para uma aplicação específica.

Embora o *hardware* tenha sido desenvolvido com uma aplicação específica em vista, as seguintes características podem ser indicadas:

- Escalável, uma vez que o ANNP pode ser usado para redes de tamanhos diferentes.
- Resolução elevada, porque as entradas e os cálculos internos são efectuados em notação de vírgula flutuante de 32 bits.
- Função de activação exacta, porque o erro máximo permitido na função activação implementada é de 0,0000218.

8.3.3 Implementação de *hardware*

Há três problemas principais com os quais é necessário lidar quando uma implementação de *hardware* com uma FPGA é considerada:

- Notação
- Função de activação
- Capacidades do dispositivo FPGA a utilizar.

A notação é um assunto chave na implementação de *hardware*. É quase consensual que a notação de vírgula flutuante deve ser utilizada se uma resolução elevada for pretendida com um número mais baixo de bits. O problema aqui é a complexidade resultante das operações necessárias (multiplicação, divisão, soma) com esta notação.

A função de activação é particularmente difícil de implementar no caso das funções sigmoidais. A implementação directa, por exemplo, da tangente hiperbólica requereria um somador, um multiplicador e uma função exponencial.

Os problemas indicados podem ser resolvidos se a capacidade do *hardware* disponível for suficiente, uma vez que são as potencialidades do dispositivo que limitam as soluções possíveis.

Notação

A notação escolhida foi a de vírgula flutuante com 32 bits de acordo com a norma IEEE 754-1985.

Embora fosse afirmado em [130] que "Algumas tentativas foram feitas para implementar RNs em FPGAs com pesos em vírgula flutuante. Apesar disso, nenhuma implementação bem sucedida foi apresentada até à data", e de [131] ter concluído que "a precisão de vírgula flutuante não é ainda praticável em RNs implementadas com FPGAs", existiam já pelo menos duas aplicações descritas que usaram a notação de vírgula flutuante. Em [127] de 17 bits e em [132] de 24 bits.

A notação de vírgula fixa requereria um número maior de bits para obter a mesma resolução e o mesmo número máximo a ser representado.

Outras soluções poderiam passar pela aritmética de *pulse stream* [126] ou de *bits-tream* [133].

Função de activação

No trabalho actual, a função de activação utilizada foi a tangente hiperbólica (equação 8.3).

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{8.3}$$

Esta equação pode ser rearranjada de modo a que tenha somente uma função exponencial mas ainda requer as operações de soma, de divisão e o cálculo da exponencial.

Percebendo que a implementação directa não é apropriada para a solução actual, a tangente hiperbólica foi estudada de forma a simplificar a sua implementação.

Como o objectivo deste trabalho estava definido desde o início, era possível estabelecer um erro máximo e usar uma aproximação clássica com uma LUT. Esta possibilidade foi testada, mas rapidamente se verificou que esta solução seria demasiado dispendiosa em termos de *hardware*.

Uma nova aproximação foi então testada: aproximação linear por secções. Isto corresponde a uma das soluções clássicas para a implementação da função da activação [130], mas neste caso com uma variante importante.

Enquanto a maioria das soluções foram feitas usando somente três secções lineares para aproximar a tangente hiperbólica, a decisão aqui foi de usar tantas secções quantas as necessárias para obter a precisão previamente definida.

Com este objectivo, foi realizado um estudo para preparar as secções lineares e para determinar quantas eram necessárias. Após ter reduzido a tangente hiperbólica à parte essencial que necessita ser representada, foi implementado o seguinte algoritmo:

```
1° Escolher a primeira secção linear.
2° Comparar esta aproximação linear com a tangente hiperbólica do
MATLAB até o erro máximo permitido ser atingido.
3° Começar uma nova secção linear.
4° Repetir a operação até que a região considerada importante esteja
completamente representada.
```

Este algoritmo conduziu à representação da tangente hiperbólica com 256 secções lineares e fornece um erro máximo de 0,0000218 nos valores de saída que estão na gama [- 1 , 1].

Este valor pode ser comparado com outras soluções como a que utilizou a série de Taylor em [127], que obteve um erro de 0,51% e a aproximação linear por secções usada em [132], que obteve 0,0254 "de desvio padrão com respeito à forma analítica" da função.

Vale a pena notar que para obter este erro com uma aproximação clássica com LUT seriam necessárias 18110 amostras da função. Estas amostras representadas na notação de 32 bits que foi utilizada requereriam mais do que uma única FPGA do tipo usado, apenas para representar a função de activação.

Plataforma de *hardware*

A plataforma de *hardware* usada é a Cyclone EP1C20F324C7 FPGA da ALTERA, no kit cyclone SmartPack da Parallax que pode ser visto na figura 8.2.

Esta FPGA é composta por 20060 elementos lógicos (do inglês *Logic Elements* - LE), 294912 bits de memória e duas *Phase Locked Loops*. O kit Cyclone SmartPack inclui um porto RS-232, um porto JTAG (*Joint Test Action Group*), 8Mbit de memória *flash* para armazenar a configuração do dispositivo e o conteúdo da memória para arranque após *power-up*, um oscilador de 50-MHz e 128 pinos de *Input/Output*.

O desenvolvimento do circuito foi feito no ambiente Quartus II.

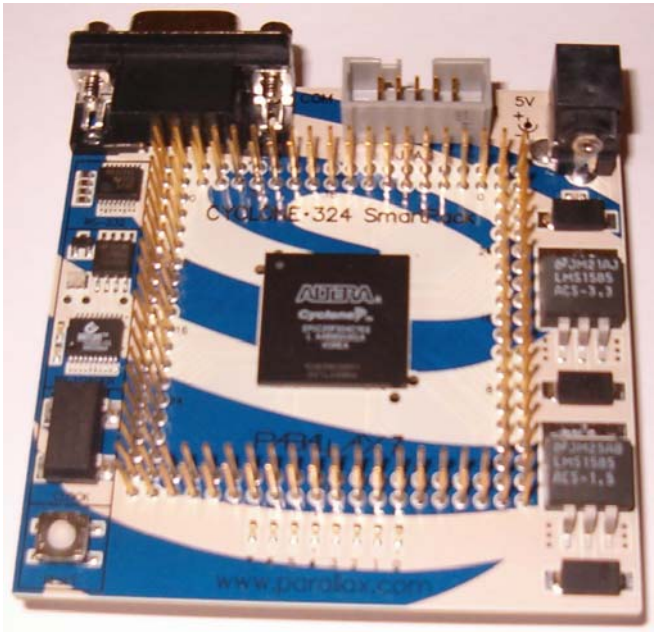


Figura 8.2: Imagem do *kit* do cyclone SmartPack.

Implementação

As potencialidades do dispositivo limitam as escolhas que podem ser feitas porque no final, o projecto escolhido tem que caber no *hardware*.

A elevada resolução escolhida limita a quantidade de *hardware* que é possível colocar na FPGA e foi decidido reduzir o *hardware* ao mínimo indispensável para implementar uma FNN.

O ANNP foi desenvolvido usando a linguagem *Very High Speed Integrated Circuits Hardware Description Language (*VHDL) e implementa os seguintes componentes:

- Multiplicador de 32 bits em notação de vírgula flutuante.
- Somador de 32 bits em notação de vírgula flutuante.
- Função de activação.

- Blocos de memória.
- Três máquinas de estados.
- Comunicação série usando o protocolo RS-232.

Os blocos de memória são usados para guardar os pesos, as entradas e os resultados parciais e finais. As três máquinas de estados são usadas para controlar a operação do ANNP e os dados de entrada e de saída através da porta RS-232.

A estrutura básica é composta por um multiplicador, um somador e uma função de activação (figura 8.3) e tem de ser usada diversas vezes para implementar a RN.

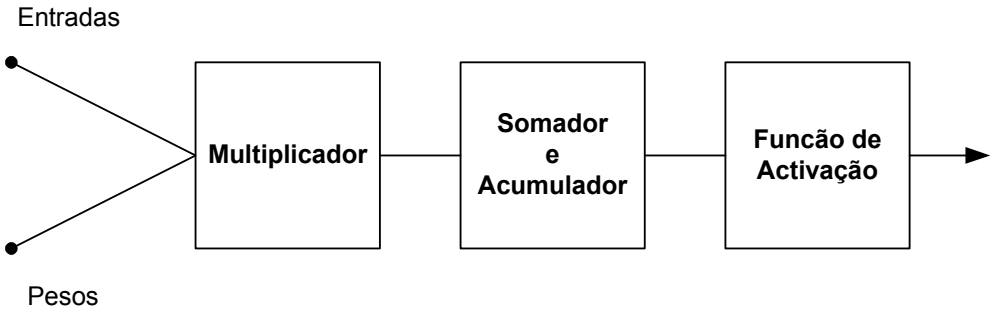


Figura 8.3: Estrutura lógica do bloco de processamento básico no ANNP

O ANNP está preparado para implementar RNs até 16 neurónios por camada e até 17 camadas.

O número de LEs usado é de 4.446, o que significa 22% do número total de LEs disponível.

A implementação da função de activação usa três ROMs de 256x32 bits, ou seja 24,576 bits, que são implementados em 6 blocos de 256x16 bits.

A implementação completa usa 235,008 bits, que são de facto 285,696 se forem contados os bits de paridade (e outros que são usados para funções especiais), de uma quantidade total de 294,912 bits disponíveis, o que significa que 96,9% dos bits foram usados (somente 2 blocos de 4096 bits não foram utilizados) e esta é a limitação real para implementar uma arquitectura maior que poderia ter sido expandida no número de blocos de processamento básico utilizado ou ampliando a capacidade funcional deste bloco.

Do total da memória, a maior parte foi utilizada para armazenar os pesos, uma vez que o ANNP reservou memória para uma quantidade máxima de 6,416 pesos, incluindo os de deslocamento.

O número máximo de entradas que o ANNP está preparado para processar é de 16.

O protocolo de comunicação série, RS-232 está configurado para funcionar a 115200 bps, uma vez que a comunicação é feita com um *start* bit e um *stop* bit, isto significa 11520 Bps.

8.3.4 Sistema de teste

A implementação de *hardware* da RN foi testada usando diversos modelos de um sistema, que foram preparados previamente em MATLAB.

Este sistema é um protótipo de um forno em escala reduzida, que está afectado por ruído de medida. Para detalhes adicionais ver [106].

Para melhorar a fase de teste e de correcção de erros, foi desenvolvido *software* que controla a operação do ANNP passo a passo e permite obter informação sobre o estado interno das variáveis e dos registos.

A janela principal desta aplicação que funciona em MATLAB pode ser vista na figura 8.4.

Esta aplicação é usada também para enviar dados e comandos à FPGA e receber os resultados. A comunicação entre MATLAB e FPGA é feita usando tramas de 48 bits que enviam comandos e dados.

A comunicação entre FPGA e MATLAB é feita usando tramas de 32 bits que contêm a resposta ao comando precedente emitido.

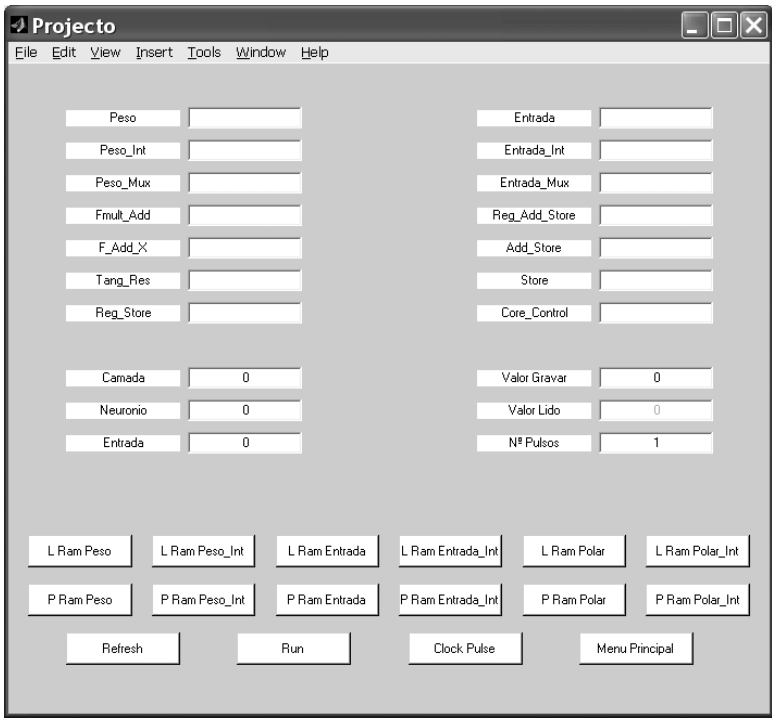


Figura 8.4: Janela principal da aplicação de teste e correcção de erros.

Com a taxa de comunicação implementada com o standard RS-232 é obtida uma velocidade de 1920 pesos por segundo, isto significa que uma rede de, por exemplo 100 pesos, levará 52 milisegundos a ser configurada.

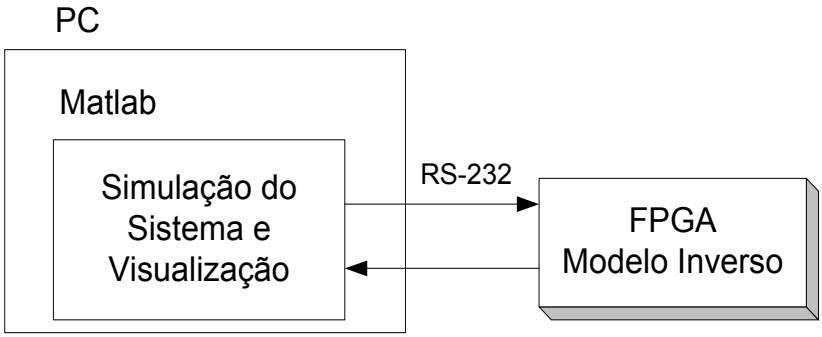


Figura 8.5: Método de teste da FPGA.

Nome do modelo	Entradas	Neurónios
Modelo 1	4	6
Modelo 2	2	3
Modelo 3	4	4

Tabela 8.8: Resumo das características dos modelos inversos testados na FPGA.

8.3.5 Resultados

Três conjuntos de modelos foram usados para testar o *hardware*. Os conjuntos consistem em pares (um modelo directo e um inverso) com número diferente de entradas e de neurónios na camada escondida. Todos os modelos têm apenas uma camada escondida com tangente hiperbólica como função de activação e função de activação linear na camada da saída.

O método escolhido para verificar a qualidade da solução desenvolvida foi implementar os modelos inversos do sistema na FPGA e os modelos directos do sistema em MATLAB, desta maneira o modelo em MATLAB poderia ser usado em substituição do sistema e seria possível implementar acções de controlo (ver figura 8.5).

A estratégia de controlo utilizada foi um simples controlo DIC, que está representado na figura 8.6.

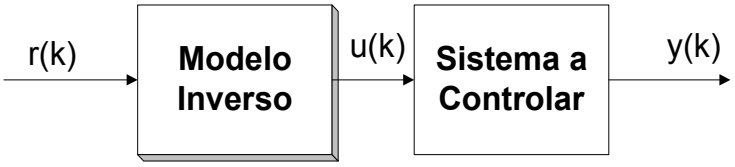


Figura 8.6: Diagrama de blocos para o controlo directo inverso.

A tabela 8.8 mostra as características dos modelos inversos testados na FPGA.

Nas figuras 8.7, 8.8 e 8.9 podem ser vistos os resultados do controlo DIC com três referências diferentes. Na figura 8.7 é utilizada uma rampa simples, na figura 8.8 a

referência é uma rampa seguida de um sinal de onda quadrada e na figura 8.9 é usado um sinal pseudo-aleatório. Estes resultados são para o conjunto de modelos designados como modelo 1.

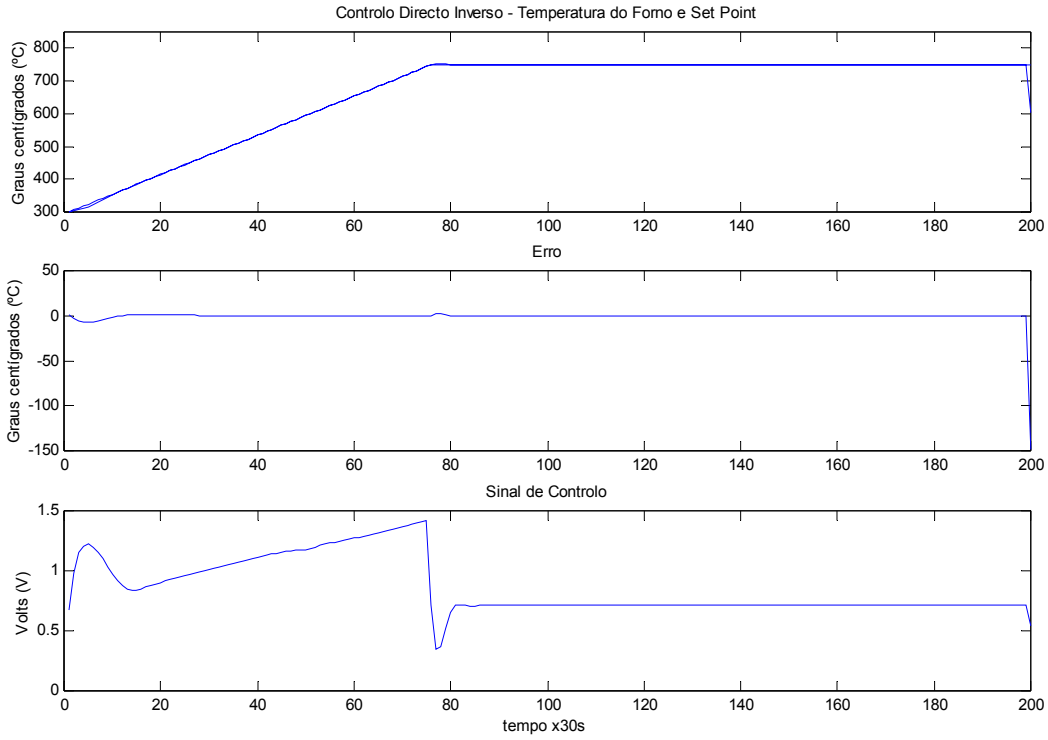


Figura 8.7: Resultados do controlo directo inverso utilizando a FPGA com o sinal da rampa.

Para completar a análise dos resultados e separar o erro que é introduzido pela utilização da FPGA do que é introduzido pelos modelos, o mesmo controlo foi implementado com ambos os modelos em MATLAB.

Os resultados são apresentados na forma de erro quadrático médio na tabela 8.9.

Como pode ser visto pelos resultados apresentados na tabela 8.9, a diferença obtida entre o controlo com a FPGA e o controlo simulado em MATLAB, em termos de erro quadrático médio, não excede $5e^{-8}$.

8.3.6 Conclusões e trabalho futuro

Este trabalho propõe uma implementação em *hardware* de uma RN usando uma FPGA. A FPGA foi escolhida por causa dos preços mais baixos que permite para uma única cópia do circuito.

O que se pretendia com este trabalho era obter uma solução de *hardware* que permitisse a utilização directa dos pesos, preparados geralmente num ambiente de

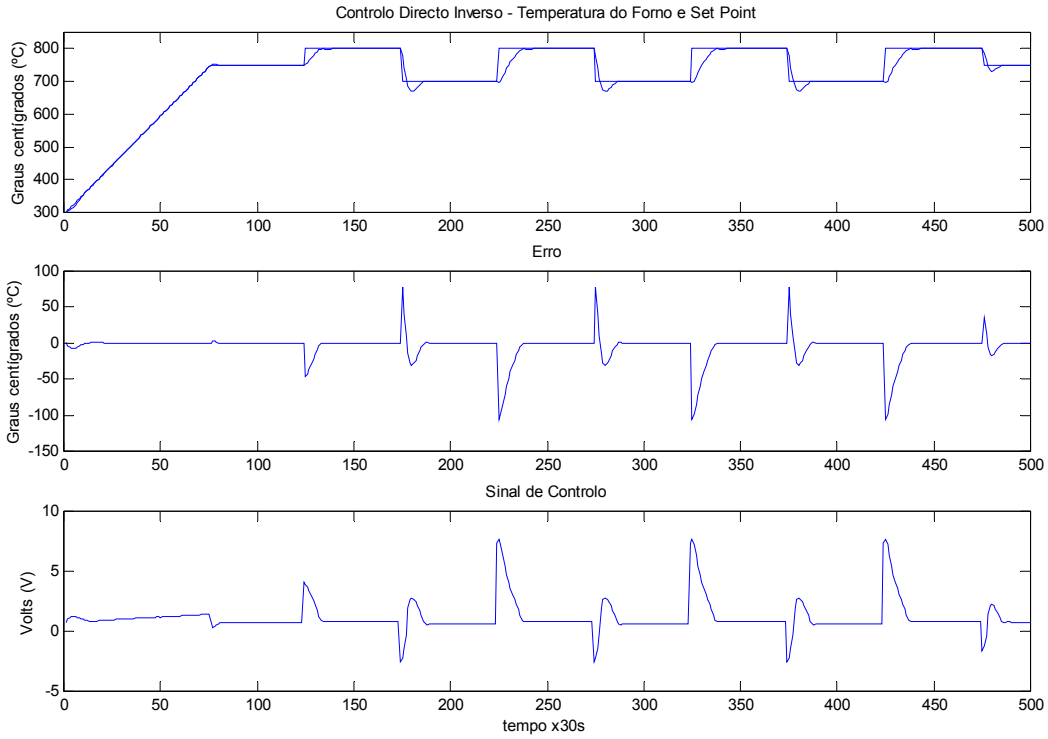


Figura 8.8: Resultados do controlo directo inverso utilizando a FPGA com o sinal de rampa e de onda quadrada.

Nome	Simulação em MATLAB			Resultados com a FPGA		
	rampa	rampa + quad.	pseudo-aleatório	rampa	rampa + quad.	pseudo-aleatório
mod.1	1,2146e^{-4}	0,0146	1,3343e^{-4}	1,2151e^{-4}	0,0146	1,3345e^{-4}
mod.2	7,8682e^{-5}	0,0040	1,3445e^{-4}	7,8666e^{-5}	0,0040	1,3440e^{-4}
mod.3	8,1468e^{-6}	0,0069	1,5426e^{-5}	8,1456e^{-6}	0,0069	1,5426e^{-5}

Tabela 8.9: Comparação entre os resultados obtidos com a FPGA e o MATLAB em termos de erro quadrático médio.

software com uma precisão muito mais elevada do que as obtidas nas implementações com FPGAs.

Este objectivo foi alcançado com sucesso, como foi demonstrado pelos testes do controlo feitos com modelos de um sistema real, onde o erro máximo obtido, usando o erro quadrático médio como medida, foi de 5e^{-8}.

Um algoritmo novo para aplicar com a aproximação por secções lineares foi apresentado e permitiu a implementação de uma tangente hiperbólica com elevada precisão. Este algoritmo permite a definição do erro máximo que é aceitável e fornece o número

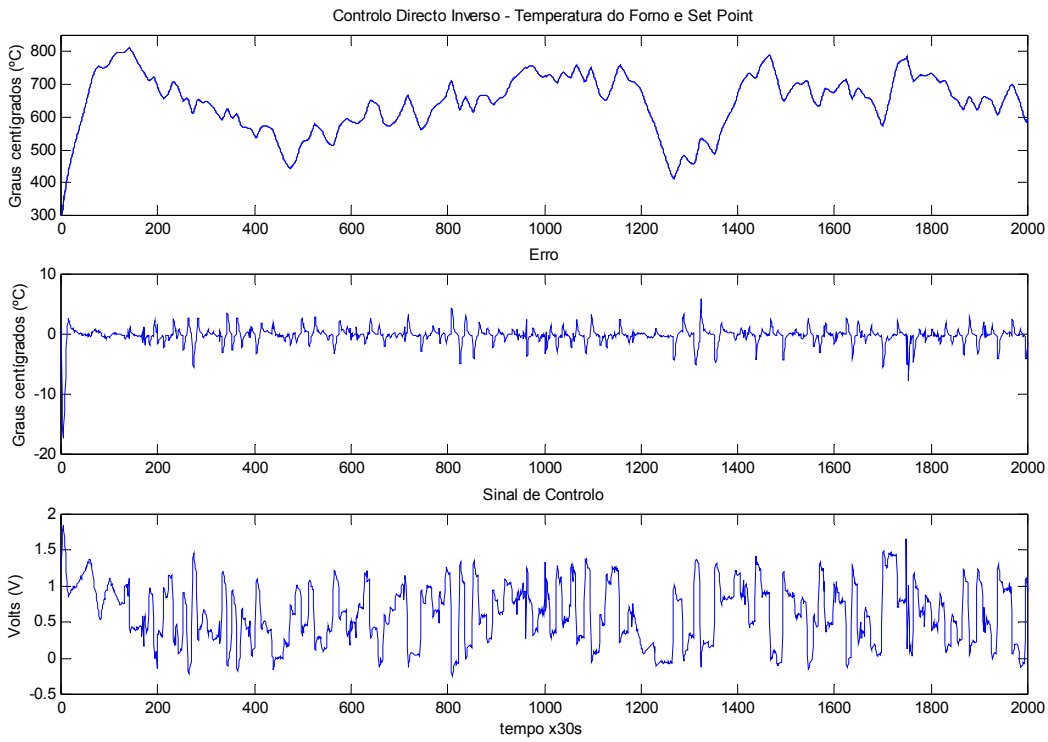


Figura 8.9: Resultados do controlo directo inverso utilizando a FPGA com o sinal pseudo-aleatório.

e as equações das secções lineares correspondentes.

Como trabalho adicional os autores gostariam de manter o objectivo que conduziu a esta implementação, ou seja a precisão elevada, e desenvolver soluções que permitam processar um neurónio completo ou uma camada completa de uma só vez. Com o aumento da densidade das FPGAs este parece ser um objectivo real no curto prazo.

8.3.7 Agradecimento

Os autores gostariam de agradecer à empresa ALTERA e ao seu programa universidade pelo apoio e pelo *software* utilizado neste trabalho.

8.4 Conclusão

Este capítulo aborda a implementação de RNs em *hardware* e está dividido em duas partes: um estudo do *hardware* comercial existente e apresentação de uma implementação que utiliza como plataforma uma FPGA.

O estudo do *hardware* comercial foi efectuado para verificar se existia no mercado uma solução que permitisse uma utilização fácil das RNs treinadas em *software*, mas

como se verificou que tal não acontecia, foi preparada uma implementação em *hardware* com uma FPGA. A FPGA foi escolhida por ser uma opção viável economicamente para circuitos dos quais serão produzidas poucas cópias.

Capítulo 9

Conclusões e trabalho futuro

"No amount of experimentation can ever prove me right; a single experiment can prove me wrong." - Albert Einstein, cientista. (1879 - 1955)

9.1 Conclusões

Este é um trabalho transversal que discute do algoritmo até à implementação das RNs aplicadas à área de controlo. São abordadas as RNs como instrumento para a área de controlo e nessa perspectiva são estudados algoritmos, formas de treino dos modelos, malhas de controlo e implementações em *hardware*.

Esta tese apresenta diversas contribuições originais e trabalha com dados reais provenientes de um sistema afectado por ruído.

Para além das contribuições salientadas:

- a técnica híbrida genérica/especializada para modelos inversos que permite a validação de uma forma similar à utilização real dos modelos.
- a malha de Controlo Aditivo Baseado em Modelo Interno que tem a vantagem de substituir o Controlador Aditivo por uma malha realimentada e que aplicada de forma genérica é bastante útil para a comutação em situações de treino *on-line.*
- a implementação do algoritmo de Levenberg-Marquardt em janela deslizante com Paragem de Treino Antecipada para treino *on-line.*
- uma solução de *hardware* baseada em FPGA cuja virtude é a elevada resolução e a qualidade da implementação da tangente hiperbólica

esta tese tem o mérito de utilizar quatro malhas de controlo distintas, com as quais foi possível obter controlo de elevada qualidade com dados de um sistema real.

9.1.1 Técnica híbrida genérica/especializada

A técnica híbrida genérica/especializada permite resolver os problemas associados às técnicas genérica e especializada na preparação dos modelos inversos. Com esta técnica existe uma garantia de que o modelo obtido é não só bom numericamente sobre a sequência de teste, como é bom para a função de controlo, o que por vezes não é conseguido, nomeadamente com técnicas de optimização, em que se obtém um modelo que é uma boa solução numérica para a sequência de teste utilizada mas que não permite obter um controlo aceitável.

9.1.2 Controlo Aditivo Baseado em Modelo Interno

A malha de Controlo Aditivo Baseado em Modelo Interno, como fica demonstrado pelos resultados conseguidos, permite obter resultados de controlo superiores aos que são alcançados com a malha que pretende substituir: a malha de Controlo Aditivo. É além disso, a estrutura geral a partir da qual é possível obter todas as outras estruturas utilizadas neste trabalho e pode ser usada como malha de comutação em situação de treino *on-line*.

9.1.3 Implementação do algoritmo Levenberg-Marquardt em janela deslizante com Paragem de Treino Antecipada

A implementação do algoritmo de Levenberg-Marquardt em janela deslizante com Paragem de Treino Antecipada permitiu resolver a dificuldade de utilizar este algoritmo *on-line*. Sendo o algoritmo de Levenberg-Marquardt reconhecidamente mais eficaz do que o bem conhecido *steepest descent*, a complexidade da sua implementação não permitiu até ao momento uma formulação iterativa. Com esta solução, que usa uma janela deslizante para permitir a utilização de uma versão *batch* do algoritmo e a Paragem de Treino Antecipada como garantia de evitar o problema do treino excessivo, é possível utilizar este algoritmo, sem limitações, para o treino *on-line*.

9.1.4 Implementação em *hardware* de uma RN com uma FPGA.

A implementação em *hardware* de uma RN utilizando como suporte uma FPGA permitiu que este trabalho percorresse todas as fases de implementação da malha de controlo, do modelo ao *hardware*. A implementação desenvolvida tem como vantagens a utilização de vírgula flutuante com elevada precisão (32 bits), o que permite o uso directo de modelos preparados nos ambientes habituais de treino das RNs e o algoritmo criado para a aproximação da tangente hiperbólica que permite escolher o número de secções lineares a utilizar em função do erro máximo permissível.

9.1.5 Qualidade de controlo

Relativamente à aplicação escolhida para testar as soluções preparadas ao longo desta tese, é possível afirmar que foi obtido controlo de elevada qualidade tanto nas imple-

mentações com treino *off-line* como com as implementações *on-line*. Na generalidade dos casos, excepção feita às fases iniciais de aquecimento e à malha PID, foi obtido um erro na temperatura inferior a um grau centígrado, apesar da não-linearidade da aplicação e do ruído existente nas leituras.

Como é expectável este trabalho abre também novos caminhos que são discutidos na secção seguinte.

9.2 Trabalho futuro

O trabalho futuro que se apresenta como provável segue desde já três linhas distintas: o melhoramento das funcionalidades do treino em tempo real, o desenvolvimento de mais soluções de *hardware* e o estudo da influência do *jitter* sobre os controladores com RNs.

9.2.1 Sistemas de tempo real

No primeiro caso existem dificuldades reconhecidas não só em fazer respeitar condições de tempo real em ambientes *Windows*, como em coordenar a execução de duas tarefas distintas como é o caso das tarefas de treino e de controlo executadas durante os processos de treino *on-line*.

Estas dificuldades podem ser superadas através da utilização do RTAI (*The Real-time Application Interface*), em ambiente Linux, em conjunto com o Matlab/Simulink ou o Scilab/Scicos. As ferramentas Scilab/Scicos [134] são *open source*, de utilização gratuita, e têm as mesmas funções do Matlab/Simulink, ou seja, o Scilab é um pacote de software científico com diversas funções matemáticas implementadas e o Scicos é o ambiente gráfico correspondente.

A ferramenta RTAI-lab [135] facilita a implementação de sistemas de controlo assistido por computador em ambientes RTAI-Linux. O RTAI-lab, que pode ser utilizado com o Matlab/Simulink ou com Scilab/Scicos, usa o código gerado por estas ferramentas e transforma-o de forma a poder ser processado pelo ambiente RTAI, por forma a poder respeitar as restrições de tempo real.

Este tipo de solução permitirá utilizar dois processos distintos a comunicarem entre si, executando um o controlo, enquanto o outro efectua o treino dos modelos. O processo que executa o controlo será também responsável pela amostragem dos sinais e fornecerá os dados ao processo que implementa o treino. Este último será responsável por, a partir dos dados que recebe, criar os modelos e fornecê-los ao processo de controlo.

No caso prático utilizado no presente trabalho foi utilizado um sistema com um período de amostragem que permitiu o treino de modelos no decurso de poucas amostragens. No entanto, o mesmo procedimento é aplicável sem nenhuma limitação a processos cujo período de amostragem seja mais baixo, desde que seja aceitável que durante alguns períodos de amostragem o controlo seja de menor qualidade, enquanto são preparados os modelos do sistema.

Teste de algoritmos *on-line*

Uma componente importante da implementação em tempo real poderá passar pela implementação de novos algoritmos ou algoritmos modificados. Neste caso assume importância testar as modificações ao algoritmo de Levenberg-Marquardt propostas em [34].

9.2.2 *Hardware*

Em termos de implementação de RNs em hardware, a solução desenvolvida representa apenas um pequeno passo em termos do que é possível realizar. Não deixa de ser um passo importante por se tratar de uma implementação em vírgula flutuante de 32 bits e pela solução encontrada para a implementação da função de activação mas, com o contínuo aumento de densidade das FPGAs, em breve será possível colocar no seu interior a totalidade do *hardware* necessário para implementar uma RN de pequena dimensão. Será importante poder construir soluções de sucessivamente maior capacidade: implementar uma camada e implementar uma rede.

Uma solução mais global para facilitar a implementação de RNs em *hardware* poderá passar pelo desenvolvimento de uma *toolbox* com soluções prontas e optimizadas com os blocos mais comuns como as funções de activação, os multiplicadores, os somadores e os acumuladores com possibilidade de escolha da resolução pretendida. Como para o mesmo fabricante os elementos base de configuração não são sempre iguais (é habitual serem-no dentro da mesma família), para obter melhores resultados, é necessário proceder à optimização para cada família de um conjunto de fabricantes.

Tolerância a falhas

Outra linha interessante de investigação diz respeito à tolerância a falhas das RNs, que tem sido apontada como uma das vantagens associadas ao processamento paralelo, mas cuja verdadeira utilidade só poderá ser verificada através da utilização de *hardware* específico para as RNs.

Neste caso é necessário avaliar o impacto que uma falha do tipo colagem a 1 ou colagem a 0 (do inglês *stuck at 1* e *stuck at 0*) produz na saída de uma RN, ou estudar outro tipo de modelização que possa ser usada para as falhas específicas das RNs (como foi proposto em [136]) e avaliar o seu impacto.

Uma vez realizado este estudo e tendo *hardware* para implementar a RN, é possível estudar as seguintes alternativas para maximizar a tolerância a falhas:

- alterar algoritmos de treino.
- estudar processos de reconfiguração da RN após uma falha.
- modificar a RN após a fase de treino sem alterar a sua arquitectura.
- modificar a RN após a fase de treino alterando a sua arquitectura.

A possibilidade de alterar algoritmos de treino para obter uma maior tolerância a falhas foi já abordada em [36], [137] e [136].

Para a implementação da segunda solução é indispensável a existência de algoritmos de treino *on-line* e de informação sobre as falhas. Esta informação pode ser obtida através de uma cadeia de *Boundary Scan* [138] alargada e de informação de teste com cobertura extensiva de forma a permitir a identificação da falha e a reorganização da RN sem o(s) elemento(s) danificados.

As restantes alternativas que abordam a possibilidade de alterar uma RN, após a fase de treino, com ou sem a alteração da arquitectura interna, por forma a torná-la mais tolerante a falhas, deverão conduzir a ferramentas que de forma autónoma efectuem a alteração da RN.

9.2.3 Influência de *jitter*

Pretende-se também investigar a adequação das RNs para controlo de sistemas sujeitos a *jitter*, procurando determinar se o seu princípio de funcionamento contribui para a obtenção de controladores mais robustos e se existem, entre as diferentes malhas de controlo e tipos de modelos, soluções mais adequadas para serem utilizadas nestas condições.

Índice

AFC
    Controlador Aditivo, 72, 73, 75, 77, 79–81, 114–116, 163, 164
AIMC
    Controlador Aditivo Baseado em Modelo Interno, 69, 77–81, 113, 115–117, 126, 130, 134, 163, 164
Algoritmos Genéticos, 61
analógicas, implementações, 143
ARMAX, 37
ART, 13
ARX, 37
ASIC, 140, 141, 151
auto-organizadas, redes, 8

Backpropagation, 8, 14, 15, 21, 25, 27, 142, 145, 148
BFGS
    Broyden-Fletcher-Goldfarb-Shanno, 25
Boltzmann, 13, 142, 148
Boundary Scan, 167

capacidade de generalizar, 51
Conjugação, 65
Controlador Aditivo misto, 74
Controlador Aditivo puro, 74
Controller Output Error Method, 49, 72
controlo automático, 5
CPPS
    Connection Primitives Per Second, 140
CPS
    Connection Per Second, 140
CPSPW
    Connection Per Second Per Weight, 140
Cruzamento, 62
CUPS
    Connection Update Per Second, 140

deleção, 65
DFP
    Davidon-Fletcher-Powell, 25
DIC
    Controlo com Modelo Inverso, 69, 71, 72, 77, 80, 81, 106, 107, 109, 114, 116, 118, 120, 126, 128, 130
digitais, implementações, 143
dithering, 127
DSP, 139, 140, 144, 145
duplicação, 65

Early Stopping, 56
Early stopping, 54, 105, 109, 113, 118
Elitismo, 63
erro de deslocamento, 52
erro de variância, 52
escalar, 32, 75, 80, 98, 127
Esqueletização, 55

FCR
    Fully Connected and Recurrent, 142
filtragem, 34
FIR, 36
Fisher, testes de, 56
FNN, 9, 13, 27, 29, 57, 140, 142, 147, 150, 154
FPGA, 138, 140, 144, 150–153, 158, 164
full custom, 151

Gauss-Newton, 15, 27
Grossberg, 8

híbridas, implementações, 147

Hopfield, 8, 13, 142, 145

IMC
Controlador Baseado em Modelo Interno, 75, 76, 78, 80, 81, 114–116, 118–120, 126, 130, 131
Implementações em Grupo, 24
implementações on-line, 121, 122
Implementações Recursivas, 24
Indirect Inverse Adaptation, 48, 72
intervalo de amostragem, 93

janela deslizante, 113, 122–124, 128, 130, 134, 135, 164
jitter, 167
JTAG
Joint Test Action Group, 154

KNN
K-Nearest Neighbour, 142, 147
Kohonen, 8, 140

Levenberg-Marquardt, 14, 15, 18, 20, 25–27, 56, 87, 102, 106, 113, 121, 122, 128, 134, 135, 163, 164, 166
LUT, 145, 146, 153

Mutação, 62

neuro-chips, 141, 149
neuro-computadores, 141, 149
neuro-fuzzy, 134
Newton, 15, 27

Optimal Brain Damage, 55
Optimal Brain Surgeon, 55
Optimização sem derivadas, 23
Output Error, 38
overfitting, 35, 56, 104
overtraining, 35, 51, 56, 57

Paragem de treino antecipada, 55, 60, 83, 100, 104, 118, 121–124, 128, 130, 163, 164
paragem de treino antecipada, 54, 56
período de amostragem, 96
PID, 165
PLD, 144
PNN
Probabilistic Neural Networks, 142, 147
poda, técnicas de, 55

Quasi-Newton, 15, 25

RBF, 13, 27, 140, 142, 144, 146, 147
RCE
Restricted Coulomb Energy, 142, 147
Regularização, 53, 55–57, 59, 60, 83, 100, 102, 104, 105, 107, 109, 119, 120
Regularização explícita, 54, 104, 107, 109
Regularização implícita, 54, 104, 107, 109
Reinforcement Learning, 13
resposta em malha aberta, 96
RISC, 139, 140, 146, 147
ROI
Region of Influence, 142, 147
RTAI, 165

Sea of Gates, 140, 147, 151
self-organizing networks, 8
SIMD, arquitectura, 142, 144–146
Sistemas de tempo real, 165
slice, arquitectura, 144
spiking neurons, 25, 27
Steepest Descent, 15–18, 21, 25–27, 56, 67, 87, 164
Systolic, arquitectura, 144, 146, 147

Técnica híbrida genérica/especializada, 30, 49, 50, 100, 104, 122, 164
Tolerância a falhas, 166
transdução, 65
treino excessivo, 51, 54, 104, 122, 124
treino, época de, 24

ULSI, 149

validação cruzada, 56
VHDL, 150, 154
VLIW
Very Long Instruction Word, 146

VLSI, 149, 151

weigh decay, 54, 104, 105
winner-take-all, 27

Bibliografia

[1] Magnus Nørgaard. *System Identification and Control with Neural Networks*. PhD thesis, Department of Automation, Technical University of Denmark, 1996.

[2] Warren S. Sarle. Frequently asked questions about neural networks. Technical report, 2000. disponível na internet em Dezembro de 2000 no grupo de discussão swnet.ai.neural-nets.

[3] J. Almeida e Costa and A. Sampaio e Melo. *Dicionário Da Língua Portuguesa - 6ª Edição*. Porto Editora, 1992.

[4] Hagan, Demuth, and Beale. *Neural Network Design*. PWS Publishing Company, 1996.

[5] António R. Damásio. *O Erro de Descartes*. Europa- América, 1994.

[6] Hans C. Andersen. *The Controller Output Error Method*. PhD thesis, Department of Computer Science and Electrical Engineering, University of Queensland, St. Lucia 4072, Australia, 1998.

[7] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop. Neural networks for control systems-a survey. *Automatica*, 28:1083–1112, 1992.

[8] Hervé Abdi. *Les Réseaux de Neurones*. Presses Universitaires de Grenoble, 1994.

[9] G. Cybenko. Approximation by superposition of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:492–499, 1989.

[10] K. Hornik, M Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[11] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[12] Eric Ronco and Peter J. Gawthrop. Neural networks for modelling and control. Technical Report csc97008, Department of Mechanical Engineering, University of Glasgow, 1997.

[13] A. Lapedes and R. Farber. How neural nets work. *Advances in Neural Information Processing Systems, Ed. D.Touretzsky, Morgan Kaufmann*, pages 442–456, 1987.

[14] G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, 1988.

[15] O. SØrensen. *Neural Networks in Control Applications*. PhD thesis, Department of Control Engineering, Institute of Electronic Systems, Aalborg University, Denmark, 1994.

[16] Eduardo D. Sontag. Feedback stabilization using two-hidden-layer nets. *IEEE Trans. Neural Networks*, 3, 1992.

[17] J.-S. R. Jang, C.-T. Sun, and E. Mizutani. *Neuro-Fuzzy and Soft Computing*. Prentice Hall, 1997.

[18] A. E. Ruano. Artificial neural networks, CSI technical report. Technical Report TR-T-01-2004, Universidade do Algarve, 2004.

[19] K. Levenberg. A method for the solution of certain problems in least squares. *Quart. Appl. Math.*, 2:164–168, 1944.

[20] D. Marquardt. An algorithm for least -squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11:431–441, 1963.

[21] António Eduardo de Barros Ruano. *Applications of Neural Networks to Control Systems*. PhD thesis, University of Wales, 1992.

[22] Martin T. Hagan and Mohammad B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5, no. 6:989–993, 1994.

[23] M. Nørgaard, O. Ravn, N. K. Poulsen, and L. K. Hansen. *Neural Networks for Modelling and Control of Dynamic Systems*. Springer, 2000.

[24] M. Nørgaard. Neural network based system identification toolbox for use with matlab, version 1.1, technical report. Technical report, Technical University of Denmark, 1996.

[25] Bruce Burton, Farrukh Kamran, Ronald G. Harley, Thomas G. Habetler, Martin A. Brooke, and Ravi Poddar. Identification and control of induction motor stator currents using fast online random training of a neural network. *IEEE transactions on industry applications*, 33, no.3:697–704, 1997.

[26] Rauf-I-Azam. *Studies on Quasi-Newton Methods for Nonsmooth Convex Optimization*. PhD thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, 1998.

[27] Berthold Ruf. *Computing and Learning with Spiking Neurons - Theory and Simulations*. PhD thesis, Institute for Theoretical Computer Science, Technische Universitat Graz, Austria, 1998.

[28] Andrés Upegui, Carlos Andrés Peña-Reyes, and Eduardo Sanchez. A hardware implementation of a network of functional spiking neurons with hebbian learning. *BioAdit - International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, 2004.

[29] Daniel Roggen, Stéphane Hofmann, Yann Thoma, and Dario Floreano. Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot. *NASA/DoD Conference on Evolvable Hardware*, pages 189–198, 2003.

[30] Guian Zhou and Jennie Si. Advanced neural-network training algorithm with reduced complexity based on jacobian deficiency. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 9, NO. 3:448–453, 1998.

[31] A. Ruano, P. Ferreira, C. Cabrita, and S. Matos. Training neural networks and neuro-fuzzy systems: A unified view. *IFAC 15th Triennial World Congress, Spain*, 2002.

[32] P. Ferreira, E. Faria, and A. Ruano. Neural network models in greenhouse air temperature prediction. *Neurocomputing*, 43, no. 1-4:51–75, 2002.

[33] Lester S. H. Ngia and Jonas Sjöberg. Efficient training of neural nets for nonlinear adaptive filtering using a recursive levenberg-marquardt algorithm. *IEEE Trans. on Signal Processing*, 48(7):1915–1927, 2000.

[34] Lester S. H. Ngia. *System Modeling Using Basis Functions and Application to Echo Cancelation*. PhD thesis, Department of Signals and Systems School of Electrical and Computer Engineering, Chalmers University of Technology, 2000.

[35] G. Lera and M. Pinzolas. Neighborhood based levenberg-marquardt algorithm for neural network training. *IEEE Transactions on Neural Networks*, 13, NO. 5:1200–1203, 2002.

[36] Salvatore Cavalieri and Orazio Mirabella. A novel learning algorithm which improves the partial fault tolerance of multilayer neural networks. *Neural Networks*, 12:91–106, 1999.

[37] Jirí Síma and Pekka Orponen. General-purpose computation with neural networks: A survey of complexity theoretic results. *Neural Computation*, 15:2727–2778, 2003.

[38] Lennart Ljung. *System Identification - Theory for the User*. Prentice Hall, 1987.

[39] Alexandre M. Mota. *Identificação de Sistemas*. DETUA, 1997.

[40] Karl Åström and Björn Wittenmark. *Computer Controlled Systems*. Prentice Hall, 1997.

[41] Ulf Holmberg. Digital control. disponível na internet em http://www.hh.se/staff/ulho/Digital/Digital.html, 2003. Halmstad University.

[42] The University of Michigan. Control tutorials for matlab - digital control example: Designing cruise control using root-locus method. disponível na internet em http://www.engin.umich.edu/group/ctm/examples/cruise/digCCRL.html.

[43] The Intelligent Systems Laboratory The Intelligent Systems Laboratory University of Guelph. Control tutorials for matlab and simulink - digital control example: Designing cruise control using root-locus method. disponível na internet em http://wolfman.eos.uoguelph.ca/ jzelek/matlab/ctms/examples/cruise/digccrl.htm.

[44] T. Söderström and P. Stoica. *System Identification*. Prentice-Hall, 1989.

[45] Y. Le Cun, I. Kanter, and S. A. Solla. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, vol. 66, n.18:2396–2399, 1991.

[46] John P. Norton. *An Introduction to Identification*. Academic Press, 1986.

[47] Lennart Ljung. *System Identification Toolbox - for Use with MATLAB, User's Guide*. 1991.

[48] Karl Aström and Bjorn Wittenmark. *Adaptive Control*. Addison Wesley, 1995.

[49] Lutz Prechelt. Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks*, 1998.

[50] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1, nº1:4–27, 1990.

[51] Jonas Sjöberg. *Non-Linear System Identification with Neural Networks*. PhD thesis, Dept. of Electrical Engineering, Linköping University, Suécia, 1995.

[52] J. Moscinski and Z. Ogonowski. *Advanced Control with Matlab and Simulink*. Ellis Horwood, 1995.

[53] D. Psaltis, A. Sideris, and A. A. Yamamura. A multilayered neural network controller. *IEEE Control Systems Magazine*, pages 17–21, 1988.

[54] M. I. Jordan and D.E. Rumelhart. Forward models: Supervised learning with a distal teacher. Ocasional paper nº 40, Center for Cognitive Science, Massachusetts Institute of Technology, 1991.

[55] Fernando Morgado Dias, Ana Antunes, and Alexandre Manuel Mota. Automating the construction of neural models for control purposes using genetic algorithms. *28th Annual Conference of the IEEE Industrial Electronics Society, Sevilha, Espanha*, 2002.

[56] Fernando Morgado Dias, Ana Antunes, and Alexandre Mota. A new hybrid direct/specialized approach for generating inverse neural models. *6th WSEAS Int. Conf. on Algorithms, Cientific Computing, Modelling and Computation (ASCOMS'04)*, 2004.

[57] Fernando Morgado Dias, Ana Antunes, and Alexandre Mota. A new hybrid direct/specialized approach for generating inverse neural models. *WSEAS Transactions on Systems*, 3, Issue 4:1521–1529, 2004.

[58] N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. *Advances in Neural Information Processing Systems, Ed. D.Touretzsky, Morgan Kaufmann*, pages 630–637, 1990.

[59] J. Sjöberg and L. Ljung. Overtraining, regularization and searching for minimum in neural networks. *Preprint IFAC Symp. on Adaptive Systems in Control and Signal Processing, Grenoble, France*, page 669:674, 1992.

[60] Y. Le Cun, J.S. Denker, and S.A. Solla. Optimal brain damage. *Advances in Neural Information Processing Systems, Denver 1989, Ed. D.Touretzsky, Morgan Kaufmann*, page 598:605, 1990.

[61] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Proceedings of NIPS 5, San Mateo, California*, page 164:172, 1993.

[62] Morten With Pederson and Lars Kai Hansen. Recurrent networks: Second order properties and pruning. *Proceedings of the Neural Information Processing Systems*, 7:673–680, 1994.

[63] Morten With Pedersen, Lars Hansen, and Jan Larsen. Pruning with generalization based weight saliencies: OBD, OBS. *Proceedings of the Neural Information Processing Systems*, 8:521–527, 1995.

[64] M. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assesment. *Advances in Neural Information Processing Systems, Ed. D.Touretzsky, Morgan Kaufmann*, 1:107–115, 1989.

[65] Asriel U. Levin, Todd K. Leen, and John E. Moody. Fast pruning using principal components. *Advances in Neural Information Processing Systems, Ed. D.Touretzsky, Morgan Kaufmann*, 6, 1994.

[66] I. T. Jollife. *Principal Component Analisys*. Spinger-Verlag, 1986.

[67] Russell Reed. Pruning algorithms - a survey. *IEEE transactions on Neural Networks*, 4, no.5:740–747, 1993.

[68] Lai-Wan Chan. Levenberg-marquardt learning and regularization. *Progress in Neural Information Processing, edited by S. Amari and L. Xu, Springer Verlag*, pages 139–144, 1996.

[69] Rich Caruana, Steve Lawrence, and C. Lee Giles. Overfitting in neural networks: Backpropagation, conjugate gradient, and early stopping. *Neural Information Processing Systems, Denver, Colorado*, 2000.

[70] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *In Advances in Neural Information Processing Systems 4, J. E. Moody, S. J. Hanson and R. P. Lippmann, eds. Morgan Kauffmann Publishers*, pages 950–957, 1995.

[71] Andreas Weigend, David Rumelhart, and Bernardo Huberman. Generalization by weight-elimination with application to forecasting. *Advances in Neural Information Processing Systems, J. E. Moody, S. J. Hanson and R. P. Lippmann, eds. Morgan Kauffmann Publishers*, pages 875–882, 1991.

[72] D. Chen and M. Hagan. Optimal use of regularization and cross-validation in neural network modeling. *International Joint Conference on Neural Networks*, Paper No. 323, 1999.

[73] Isabelle Rivals and Léon Personnaz. A statistical procedure for determining the optimal number of hidden neurons of a neural model. *Second International ICSC Symposium on Neural Computation, Berlin, Germany*, 2000.

[74] Ulrich Anders and Olaf Korn. Model selection in neural networks. *Neural Networks*, 12:309–323, 1999.

[75] D. Whitley. Genetic algorithms in engineering and computer science. editado por J. Periaux and G. Winter,John Wiley and Sons, 1995. capítulo em Genetic Algorithms and Neural Networks.

[76] Anabela Borges Simões. Transposiçao: Estudo de um novo operador genético inspirado biologicamente. Technical report, Tese de Mestrado no Departamento de Engenharia Informática, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, 1999.

[77] J.-S. Yang and M. L. West. A case study of PID controller tuning by genetic algorithm. *Proceedings of the IASTED International Conference on Modelling Identification and Control, Innsbruck*, 2001.

[78] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, MIT Press, Cambridge, MA, 1992.

[79] G. Bartlett. *Genie a First GA Capítulo de Practical Handbook of Genetic Algorithms, Applications*, volume I, pp31-56. CRC Press, 1995.

[80] Ajith Abraham and Baikunth Nath. Optimal design of neural nets using hybrid algorithms. *PRICAI 2000*, pages 510 – 520, 2000.

[81] J. Schaffer, D. Whitley, and L. Eshelman. Combination of genetic algorithms and neural networks: The state of the art. *Combination of Genetic Algorithms and Neural Networks, IEEE Computer Society*, 1992.

[82] X. Yao. Evolutionary artificial neural networks. *International Journal of Neural Systems*, 4:203–222, 1993.

[83] T. Hussain. Methods of combining neural networks and genetic algorithms. Tutorial Presentation, ITRC/TRIO Researcher Retreat, Kingston, Ontario, Canada, 1997.

[84] P. Köhn. Genetic encoding strategies for neural networks. *Information Processing and Management of Uncertainty in Knowledge-based Systems, Granada, Spain*, 1996.

[85] K. J. Hunt and D. Sbarbaro. Neural networks for nonlinear internal model control. *IEE Proceedings-D*, vol.138, no.5:431–438, 1991.

[86] G. Lightbody and G. W. Irwin. Nonlinear control structures based on embedded neural system models. *IEEE transactions on Neural Networks*, vol.8, no.3, 1997.

[87] C.E. Garcia and M. Morari. Internal model control: 1. a unifying review and some new results. *Ind. Eng. Chem. Process Res. Dev.*, 21:308–323, 1982.

[88] G. C. Economou, M. Morari, and B. O. Palsson. Internal model control. 5. extension to nonlinear systems. *Ind. Eng. Chem. Process Des. Dev.*, 25:403–411, 1986.

[89] William S. Levine. *The Control Handbook*. CRC Press, 1995.

[90] Fernando Morgado Dias, Ana Antunes, and Alexandre M. Mota. Additive internal model control: An application with neural models in a kiln. *28th Annual Conference of the IEEE Industrial Electronics Society, Sevilha, Espanha*, 2002.

[91] *HP34970A Data Aquisition User's Guide*. Hewlett Packard, 1997.

[92] M. Nørgaard. Neural network based control system design toolkit for use with matlab, version 1.1, technical report. Technical report, Technical University of Denmark, 1996.

[93] *MATLAB - External Interface Guide*. Mathworks, 1996.

[94] *MATLAB - Reference Guide*. Mathworks, 1996.

[95] Fernando Morgado Dias, Ana Antunes, and Alexandre Manuel Mota. Regularization versus early stopping: A case study with a real system. *2nd IFAC Conference Control Systems Design, Bratislava, República Eslovaca*, 2003.

[96] Fernando Morgado Dias, Ana Antunes, José Vieira, and Alexandre Manuel Mota. Implementing the levenberg-marquardt algorithm on-line: A sliding window approach with early stopping. *2nd IFAC Workshop on Advanced Fuzzy/Neural Control*, 2004.

[97] António E. Ruano and Ana B. Azevedo. B-splines neural network assisted PID autotuning. *International Journal of Adaptive Control and Signal Processing*, 13, Issue 4:291 – 306, 1999.

[98] António E. Ruano. Recent developments in neural network PID autotuning. no livro Computational Intelligence in Systems and Control Design and Applications - Kluwer, 2000.

[99] Fernando Morgado Dias and Alexandre Manuel Mota. A comparison between a PID and internal model control using neural networks. *5th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, EUA*, V:268–273, 2001.

[100] José António Barros Vieira, Fernando Morgado Dias, and Alexandre Manuel Mota. Comparison between artificial neural networks and neurofuzzy systems in modeling and control: A case study. *SICICA - 5th IFAC International Symposium on Inteligent Components and Instruments for Control Applications*, 2003.

[101] José Vieira, Fernando Morgado Dias, and Alexandre Mota. Artificial neural networks and neuro-fuzzy systems for modelling and controlling real systems: A comparative study. *Engineering Applications of Artificial Intelligence, IFAC*, 17/3:265–273, 2004.

[102] F. M. Dias, A. Antunes, and A. M. Mota. Commercial hardware for artificial neural networks: A survey. *SICICA - 5th IFAC International Symposium on Inteligent Components and Instruments for Control Applications, Aveiro*, 2003.

[103] Fernando Morgado Dias, Ana Antunes, and Alexandre Mota. Artificial neural networks: A review of commercial hardware. *Engineering Applications of Artificial Intelligence, IFAC*, 17/8:945–952, 2004.

[104] Pedro Ferreira, Pedro Ribeiro, Ana Antunes, and Fernando Morgado Dias. Artificial neural networks processor - a hardware implementation using a FPGA. *Field-Programmable Logic and its Applications*, 2004.

[105] M. Minsky and S. Papert. *Perceptrons*. Cambridge, MA: MIT Press„ 1969.

[106] Fernando Morgado Dias and Alexandre Manuel Mota. Comparison between different control strategies using neural networks. *9th Mediterranean Conference on Control and Automation - Dubrovnik, Croatia*, 2001.

[107] Y. Liao. Neural networks in hardware: A survey. disponível na internet em http://wwwcsif.cs.ucdavis.edu/ liaoy/research/NNhardware.pdf.

[108] C. Lindsey and T. Lindblad. Review of hardware neural networks: A user's perspective. *Proceeding of 3rd Workshop on Neural Networks: From Biology to High Energy Physics, Isola d'Elba, Italy*, 1994.

[109] M. Holler. VLSI implementation of learning and memory systems: A review. *Advances in Neural Information Processing Systems 3, Morgan Kaufmann, San Mateo, Ca.*, 1991.

[110] E. Van Keulan, S.Colak, H. Withagen, and H. Hegt. Neural networks hardware performance criteria. *Proceedings of the IEEE Conference on Neural Networks*, III:1885–1888, 1994.

[111] R. Schüffny, A. Graupner, and J. Schreiter. Hardware for neural networks. *4th International Workshop Neural Networks in Applications, Magdeburg, Germany*, 1999.

[112] I. Aybay, S. Çetinkaya, and U. Halici. Classification of neural network hardware. *Neural Network World, IDG Co.*, 6, Nº1:11–29, 1996.

[113] D. Caviglia. NeuroNet roadmap - preliminary draft, part a) hardware implementations. disponível na internet em http://www.kcl.ac.uk/neuronet/about/roadmap/hardware.html.

[114] J. N. H. Heemskerk. *Overview of Neural Hardware. Neurocomputers for Brain-Style Processing. Design, Implementation and Application.* PhD thesis, Unit of Experimental and Theoretical Psychology, Leiden University, the Netherlands, 1995.

[115] Silicon Recognition INC. ZISC78 datasheet, 2002.

[116] D. Baratta, G. M. Bo, D. D. Caviglia, F. Diotalevi, and M. Valle. Microelectronic implementation of artificial neural network. *Invited paper in the proceedings of the 5th Electronics Devices and Systems International Conference, Brno, Czech Republic*, 1998.

[117] C. Lindsey. Neural networks in hardware: Architectures, products and applications. disponível na internet em http://www.particle.kth.se/ lindsey/HardwareNNWCourse/home.html.

[118] P. Ienne and G. Kuhn. Digital systems for neural networks. *", P. Papamichalis and R. Kerwin, editors, Digital Signal Processing Technology, volume CR57, of Critical Reviews Series, SPIE Optical Engineering, Orlando, Fla.*, pages 314–345, 1995.

[119] NC3001 TOTEM - a digital processor for neural networks - product description. disponível na internet em http://www.neuricam.com, 1998.

[120] NC3003 TOTEM digital processor for neural networks - data sheet rel., 1999.

[121] RC Module. NM6403 digital signal processor, data sheet v1.0.

[122] NEuroNet Research Committee. A european strategy for neural networks. disponível na internet em http://www.kcl.ac.uk/neuronet/about/roadmap/hardware.html.

[123] SAND NeuroChip - infosheet, 1997.

[124] J. Alspector, T. Jayakumar, and S. Luna. Experimental evaluation of learning in a neural microsystem. *Advances in Neural Information Processing Systems*, 4:871–878, 1992.

[125] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, 1987.

[126] P. Lysaght, J. Stockwood, J. Law, and D. Girma. Artificial neural network implementation on a fine-grained FPGA. *Field-Programmable Logic, Springer Verlag, R. W. Hartenstein, M. Z. Servit (Eds.), Czech Republic*, pages 421–431, 1994.

[127] M. A. Arroyo Leon, A. Ruiz Castro, and R. R. Leal Ascenccio. An artificial neural network on a field programmable gate array as a virtual sensor. *Proceedings of The Third International Workshop on Design of Mixed-Mode Integrated Circuits and Applications, Puerto Vallarta, Mexico*, pages 114–117, 1999.

[128] M. Skrbek. Fast neural network implementation. *Neural Network World*, 9, nº5:375–391, 1999.

[129] D. F. Wolf, R. A. F. Romero, and E. Marques. Using embedded processors in hardware models of artificial neural networks. *V Simposio Brasileiro de automação inteligente, Brasil*, 2001.

[130] J. H. Zhu and Peter Sutton. FPGA implementations of neural networks - a survey of a decade of progress. *13th International Conference on Field Programmable Logic and Applications, Lisbon*, 2003.

[131] K. Nichols, M. Moussa, and S. Areibi. Feasibility of floating-point arithmetic in FPGA based artificial neural networks. *CAINE, San Diego California*, pages 8–13, 2002.

[132] J. L. Ayala, A. G. Lomeña, M. López-Vallejo, and A. Fernández. Design of a pipelined hardware architecture for real-time neural network computations. *IEEE Midwest Symposium on Circuits and Systems, USA*, 2002.

[133] S. L. Bade and B. L. Hutchings. FPGA-based stochastic neural networks - implementation. *IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA*, pages 189–198, 1994.

[134] Scilab e scicos. disponível na internet em http://scilabsoft.inria.fr/.

[135] R. Bucher and L. Dozio. CACSD under RTAI linux with rtai-lab. *Realtime Linux Workshop, Valencia*, 2003.

[136] H. Elsimary, S.Mashali, and S.Shaheen. Performance evaluation of a novel fault tolerance training algorithm. *proceedings of the World congress on computational intelligence*, 2:856–860, 1994.

[137] H. Elsimary, S.Mashali, and S.Shaheen. Generalization ability of fault tolerant feed forward neural nets. *proceedings of the IEEE International conference on systems, man and cybernetics*, 1:30–34, 1995.

[138] Keneth Parker. *The Boundary Scan Handbook*. 1992.

Anexo A - Glossário de termos e abreviaturas

AFC	*Additive Feedforward Control*
AG	Algorítmo Genético
AIMC	*Additive Internal Model Control*
ALU	*Arithmetic Logic Unit*
ANNP	*Artificial Neural Network Processor*
ARMAX	*AutoRegressive Moving Average eXogenous signal*
ART	*Adaptive Resonance Theory*
ARX	*AutoRegressive eXogenous signal*
ASIC	*Application Specific Integrated Circuits*
BFGS	Broyden-Fletcher-Goldfarb-Shanno
COEM	*Controller Output Error Method*
CPPS	*Connection Primitives Per Second*
CPS	*Connection Per Second*
CPSPW	*Connection Per Second Per Weight*
CUPS	*Connection Update Per Second*
DFP	Davidon-Fletcher-Powell
DIC	*Direct Inverse Control*
DL	*Data Logger*
DSP	*Digital Signal Processors*
EQ	Erro Quadrático
EQM	Erro Quadrático Médio
FCR	*Fully Connected and Recurrent*
FIR	*Finite Impulse Response*
FNN	*Feedforward Neural Networks*
FP	*Floating Point*
FPGA	*Field Programmable Gate Array*

GP *General Purpose*

IIA *Indirect Inverse Adaptation*
IMC *Internal Model Control*
IP Índice de Performance

JTAG *Joint Test Action Group*

KNN *K-Nearest Neighbour*

LE *Logic Elements*
LUT *Look Up Table*

MIMO *Multi Input Multi Output*
MISO *Multi Input Single Output*

NA *Not Available*
NNARMAX *Neural Network AutoRegressive Moving Average eXogenous signal*
NNARX *Neural Network AutoRegressive eXogenous signal*
NNFIR *Neural Network Finite Impulse Response*
NNOE *Neural Network Output Error*

OBS *Optimal Brain Surgeon*
OBD *Optimal Brain Damage*
OE *Output Error*

PC *Personal Computer*
PE *Processing Elements*
PID *Proportional Integral Diferential*
PLD *Programmable Logic Device*
PNN *Probabilistic Neural Networks*
PWM *Pulse Width Modulation*

RBF *Radial Basis Function*
RCE *Restricted Coulomb Energy*
RISC *Reduced Instruction Set Computers*
RN Rede Neuronal
ROI *Region of Influence*
RTAI *The Realtime Application Interface*

SIMD *Single Instruction Multiple Data*
SCPI *Standard Commands for Programmable Instruments*

ULSI *Ultra Large Scale Integration*

VHDL *Very High Speed Integrated Circuits Hardware Description Language*
VLSI *Very Large Scale Integration*
VLIW *Very Long Instruction Word*

Anexo B - Exemplo de código de implementação de uma Rede Neuronal

Neste anexo é apresentado um exemplo de código de implementação de uma RN em Matlab e um exemplo da sua utilização numa simulação de Controlo com Modelo Inverso.

A função seguinte foi criada para calcular a saída de uma RN, desde que lhe sejam fornecidas as variáveis $NetDef$, $W1$, $W2$ e inputs, e devolve a resposta y.

```
function y=nnoutput(NetDef,W1,W2,inputs)
% Esta função calcula a saída de uma Rede Neuronal desde
% que lhe seja fornecido o vector de entradas do tipo
% [y(t-m); ...  ;y(t-1);u(t-n); ...  ;u(t-1)], o vector de
% configuração da RN do tipo NetDeff = ['HHLH';'L - - -'];
% onde H representa uma tangente hiperbólica e
% L uma função linear e as matrizes de pesos da primeira
% camada W1 e da segunda W2
% A função está desenvolvida de forma a ser utilizada em
% ciclos, uma vez que recebe apenas um vector de entrada
% gera apenas a resposta correspondente a uma amostra.
% são consideradas apenas redes com uma camada escondida
% --------- Inicializações --------
% procurar funções de activação lineares na camada escondida
L_hidden = find(NetDef(1,:)=='L')';
% procurar funções de activação com tangente hiperbólica
% na camada escondida
H_hidden = find(NetDef(1,:)=='H')';
% procurar funções de activação lineares na camada de saída
L_output = find(NetDef(2,:)=='L')';
% procurar funções de activação com tangente hiperbólica
% na camada de saída
H_output = find(NetDef(2,:)=='H')';
%----------Calcular saída---------------
% Primeira camada
```

```
h1=W1*[inputs;1];
y1=zeros(size(h1));
y1(H_hidden)=pmntanh(h1(H_hidden));
y1(L_hidden) = h1(L_hidden);
% Segunda camada
h2=W2*[y1;1];
y(L_output) = h2(L_output);
```

Fazendo uso desta função é possível implementar, por exemplo, uma simulação de Controlo com Modelo Inverso.

O código apresentado é apenas uma das formas possíveis de implementação.

```
function [y,u]=invcontr(Y,dirmodel,invmodel);
% Esta função simula Controlo com Modelo Inverso, a partir
% dos modelos contidos nos ficheiros dirmodel e invmodel e
% do sinal de referência Y
% A aproximação inteira do tempo morto máxima considerada
% é de duas amostras.
string='load ';
dirmodel=[string dirmodel];
invmodel=[string invmodel];
eval(dirmodel);
eval(invmodel);
aux=length(Y);

for i=1:aux-NNi(3)
 iinputs=[];finputs=[];
 % Calcular entradas do modelo directo
 % valores anteriores de y
 for j=1:NNf(1)
   if (i-j)>0
     finputs=[finputs;y(1,i-j)];
   else
     finputs=[finputs;0];
   end
 end
 % valores anteriores de u
 for j=0:NNf(2)-1
   if (i-j-NNf(3))>0
     finputs=[finputs;u(1,i-j-NNf(3))];
   else
     finputs=[finputs;0];
   end
 end
```

```
%Calcular a saída do modelo directo
ynew=nnoutput(NetDeff,W1f,W2f,finputs);
y(i)=ynew;
% Calcular entradas do modelo inverso
 % Sinal de referência
iinputs=[iinputs;Y(i+NNi(3))];
% valores anteriores de y e da referência
for j=1:NNi(1)
  if (i-j+NNi(3))<1
    iinputs=[iinputs;0];
    elseif (NNi(3)-j)>0
    % esta amostra ainda não existe tem de ser
    % substituída pela referência
    iinputs=[iinputs;Y(i+NNi(3)-j)];
  else
    iinputs=[iinputs;y(1,i-j+NNi(3))];
  end
end
% valores anteriores de u
for j=1:NNi(2)-1
  if (i-j)>0
    iinputs=[iinputs;u(1,i-j)];
  else
    iinputs=[iinputs;0];
  end
end
%Calcular a saída do modelo inverso
unew=nnoutput(NetDefi,W1i,W2i,iinputs);
u(i)=unew;
end
```

Anexo C - Lista de publicações

Publicações em revistas:

- "Artificial Neural Networks: a Review of Commercial Hardware", Fernando Morgado Dias, Ana Antunes, Alexandre Mota, Engineering Applications of Artificial Intelligence, IFAC, Vol 17/8, pp 945-952, 2004.
- "Artificial neural networks and neuro-fuzzy systems for modelling and controlling real systems: a comparative study", José Vieira, Fernando Morgado Dias, Alexandre Mota, Engineering Applications of Artificial Intelligence, IFAC, Vol 17/3, pp 265-273, 2004.
- "A new hybrid direct/specialized approach for generating inverse neural models", Fernando Morgado Dias, Ana Antunes, Alexandre Mota, WSEAS Transactions on Systems, Issue 4, Vol 3, pp 1521-1529, Junho de 2004.
- "Neuro-Fuzzy Systems: A Survey", José Vieira, Fernando Morgado Dias, Alexandre Mota, WSEAS Transactions on Systems, Issue 2, Vol. 3, pp. 414-419, Abril de 2004.
- "Influence of the sampling period in the performance of a real-time distributed system under jitter conditions", Ana Antunes, Fernando Morgado Dias, Alexandre Mota, WSEAS Transactions on Communications, Issue 1, Volume 3, pp. 248-253, Janeiro de 2004.
- "Geração de Vectores de Teste por Emparelhamento", Fernando Morgado Dias, Mohamed Hedi Touati, Meryem Marzouki e António Ferrari, Revista do Departamento de Electrónica da Universidade de Aveiro, 1996.

Publicações em conferências:

- "Artificial Neural Networks Processor - a Hardware Implementation using a FPGA", Pedro Ferreira, Pedro Ribeiro, Ana Antunes, Fernando Morgado Dias, Field-Programmable Logic and its Applications, Setembro de 2004.
- "Implementing the Levenberg-Marquardt Algorithm on-line: a Sliding Window Approach with Early Stopping", Fernando Morgado Dias, Ana Antunes, José Vieira, Alexandre Manuel Mota, 2^{nd} IFAC Workshop on Advanced Fuzzy/Neural Control, Setembro de 2004.

- "Influence of the sampling period in the performance of a real-time distributed system under jitter conditions", Ana Antunes, Fernando Morgado Dias, Alexandre Mota, 6^{th} WSEAS Int. Conf. on Telecommunications and Informatics (TELE-INFO'04), Cancun, México, Maio de 2004.

- "A new hybrid direct/specialized approach for generating inverse neural models", Fernando Morgado Dias, Ana Antunes, Alexandre Mota, 6^{th} WSEAS Int. Conf. on Algorithms, Cientific Computing, Modelling and Computation (ASCOMS'04), Cancun, México, Maio de 2004.

- "Neuro-Fuzzy Systems: A Survey", José Vieira, Fernando Morgado Dias, Alexandre Mota, 5^{th} WSEAS NNA International Conference on Neural Networks and Applications, Udine, Italia, Março 2004.

- "Regularization versus early stopping: a case study with a real system", Fernando Morgado Dias, Ana Antunes, Alexandre Manuel Mota, 2^{nd} IFAC Conference Control Systems Design (CSD'03), Bratislava, República Eslovaca, 2003.

- "Commercial Hardware for Artificial Neural Networks: a Survey", Fernando Morgado Dias, Ana Antunes, Alexandre Manuel Mota, SICICA - 5^{th} IFAC International Symposium on Inteligent Components and Instruments for Control Applications, Aveiro, 2003.

- "Comparison between Artificial Neural Networks and Neurofuzzy Systems in Modeling and Control: a Case Study", José António Barros Vieira, Fernando Morgado Dias, Alexandre Manuel Mota, SICICA - 5^{th} IFAC International Symposium on Inteligent Components and Instruments for Control Applications, Aveiro, 2003.

- "Automating the Construction of Neural Models for Control Purposes using Genetic Algorithms", Fernando Morgado Dias, Ana Antunes, Alexandre Manuel Mota, 28^{th} Annual Conference of the IEEE Industrial Electronics Society, Sevilha, Espanha, 2002.

- "Additive Internal Model Control: an Application with Neural Models in a Kiln", Fernando Morgado Dias, Ana Antunes, Alexandre Manuel Mota, 28^{th} Annual Conference of the IEEE Industrial Electronics Society, Sevilha, Espanha, 2002.

- "Control of a Kiln using Neuro Fuzzy Techniques", José António Barros Vieira, Fernando Morgado Dias, Alexandre Manuel Mota, 5^{th} Portuguese Conference on Automatic Control, Aveiro, em Setembro de 2002.

- "CAN-based Real Time Adaptive Distributed Control", Ana Antunes, Fernando Morgado Dias, Alexandre Manuel Mota, 8^{th} International CAN Conference, Las Vegas, USA, 2002.

- "A Comparison between a PID and Internal Model Control using Neural Networks", Fernando Morgado Dias e Alexandre Manuel Mota, 5^{th} World Multi-Conference on Systemics, Cybernetics and Informatics, volume V, pp.268-273, Orlando, EUA, 2001.

- "Comparison between different Control Strategies using Neural Networks", Fernando Morgado Dias e Alexandre Manuel Mota, 9^{th} Mediterranean Conference on Control and Automation, Dubrovnik, Croácia, 2001.

- "Additive Feedforward Control of a Kiln Using Neural Networks", Fernando Morgado Dias e Alexandre Manuel Mota, IASTED International Conference on Modelling, Identification, and Control, Innsbruck Austria, 2001.

- "Direct Inverse Control of a Kiln", Fernando Morgado Dias e Alexandre Manuel Mota, 4^{th} Portuguese Conference on Automatic Control, Guimarães, 2000.